AFIT/GCS/ENG/93M-01

OBJECT-ORIENTED DATABASE ACCESS FROM ADA

THESIS

Li Chou
Lt Col, ROCAF

AFIT/GCS/ENG/93M-01

AFIT/GCS/ENG/93M-01

OBJECT-ORIENTED DATABASE ACCESS FROM ADA

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Li Chou, B.S.

Lt Col, ROCAF

March 1993

Approved for public release; distribution unlimited

Table of Contents

List of Figures

List of Tables

Acknowledgements

Many persons helped me go through studying here. It is hard to name all of them, but I must mention Maj. Mark A. Roth. I owe him much for his advise and patience in helping me through this difficult time. If I am a dragon, I could be swimming in shallow water; language barrier constrains my entire space. Without Maj. Roth's endless advise and patient communication–a real life example of interfacing in languages, I would probably pack my luggage and return to my domicile before I could finish my thesis.

It is exciting to count the days preparing to go back home. For the 21 months I stay, I need to thank all my friends here; I need to thank my wife Susan (Fen-Ming) and my children Amy (Chia-Hui), Jenny (Chia-June), and Charlie (Chia-Chun). Without their support, I could not have completed this work.

Li Chou

Abstract

Ada embodies many modern software engineering principles, namely, modifiability, efficiency, reliability, and understandability. Its powerful data abstraction allows programmers to easily model objects in the real world. However, Ada does not provide data management facilities as a database management system (DBMS) does. A DBMS provides long term storage. It provides a convenient and efficient environment to manipulate data. Currently, with Ada, access to a DBMS is typically done through the use of a language extension and a preprocessor to convert the extensions to library calls appropriate for the DBMS. These systems currently support relational DBMS's and some variant of the SQL data manipulation language. However, the data structures in traditional DBMS and in Ada are very different and cause limitations that affect Ada's ability to access traditional DBMS for more complex applications, such as computer-aided engineering design.

Now, object-oriented design (OOD) is a new way of thinking about problems using models organized around real-world concepts. Currently, the OOD methodology has been implemented in object-oriented programming languages (OOPL) and object-oriented database systems (OODBMS). They provide the same methodology to handle objects. An OODBMS includes most benefits of a relational DBMS and, in addition, provides the capability to manipulate complex, heterogeneous data. ObjectStore is an OODBMS. An interface from Ada to ObjectStore could fulfill the requirements for complex applications.

To approach this research, first, the parallel data types in C and in Ada were implemented. Then the interface functions were implemented according to the functions described in the ObjectStore Reference Manual. For reasons of simplicity, the interface is done via the ObjectStore C library. Performance testing is accomplished by comparing the differences between Ada/ObjectStore and C/ObjectStore.

Ada/ObjectStore performed better in CPU time than C/ObjectStore. However, there is not much difference between Ada/ObjectStore and C/ObjectStore. The main factors that affect the result of performance still depend on the two languages own abilities. It is clear that Ada/ObjectStore provides the capability of data persistence to Ada. This result favorably affects program length, program development time, program maintainability, and application reliability.

OBJECT-ORIENTED DATABASE ACCESS FROM ADA

I. Introduction

1.1 Background

Historically, persistence of data was accomplished through file systems. File systems provide data storage required by the application programmer to manage data, but with file systems it is difficult to support data consistency, concurrency, and sharing work between applications.

Unlike file systems, database management systems (DBMS) provide an environment which is both convenient and efficient for computer applications to manipulate data. Database management systems not only support data storage, but also maintain data protection. DBMS protect data against inconsistency through concurrency control and recovery methods in spite of multiple users and system failures. In addition, DBMS provide services for data sharing, security, and query.

However, the relational database management system (RDBMS), currently the DBMS of the choice, has many shortcomings. In particular, it has deficiencies for complex design and engineering applications. The object-oriented DBMS (OODBMS) is a new generation of DBMS which incorporates the object-oriented programming paradigm into a database system. It retains the capabilities of traditional DBMSs; plus, it supports complex data types, multiple versions, and long transactions. Table 1.1 is a good summary comparing RDBMS and OODBMS support of several characteristics of engineering design tools. From this table it is evident that a good OODBMS can potentially provide the database support necessary for a complex application (11).

Ada is the standard US DoD programming language. Access to a DBMS from Ada is typically done through language extension and a preprocessor to convert the extensions to library calls appropriate for the DBMS. These systems currently support RDBMSs and some variant of the SQL data manipulation language. Current OODBMSs are written primarily for the C or C++ language. Ada can interface with C via the Ada interface facility pragma `INTERFACE`. The pragma `INTERFACE` specifies the other language and informs the compiler that an object module will be supplied for the corresponding subprogram (1). AFIT has an OODBMS, ObjectStore, an OODBMS from Object Design, Inc., and several Ada compilers. This thesis demonstrates that Ada can access an OODBMS that provides extended capabilities for Ada that has object management facilities.

Characteristic	Design Tool Example	Traditional DBMS	Object-Oriented DBMS
Complex State	References to subcomponents within circuit.	A key is required for each sub-component. Joins are required to merge into a single object.	Fundamental to the object-oriented paradigm.
Inheritance	New adder inherits attributes of a typical adder and modifies them to fit a particular circuit.	Complete specification of the schema must be defined *a priori*.	Fundamental to the object-oriented paradigm.
Complex Data Types	Graphical representation of a circuit.	Only supports basic data types such as integer and character.	Supports graphical and textual data and allows user to define data types.
Multiple Views	Top level view of design or more detailed look at a sub-component.	Must be defined in the application. Limited by record oriented retrieval.	Can be specified as a method for the object. Data is more easily retrieved using object-oriented storage techniques.
Multiple Versions	Current and historical versions.	May support multiple versions of individual records.	Generally built in as a tree structure with root node representing a version. Tree includes all objects which make up the version.
Phased Development	Top down design.	Not supported. Entire schema must be defined *a priori*.	Refined schema can inherit characteristics of a higher level and modify for next phase.
Large Data Volume	Thousands of sub-components in a circuit.	Limited only by physical storage; however, record-oriented storage may limit the size of record, causing multiple record retrievals for a single object.	Clustering by object reduces the number of disk accesses. Complex data types remove object size restrictions.
Long Transaction Duration	Designer takes two weeks to modify a specific circuit design.	Built around short business transactions. Inefficiency and failure occur with long transactions.	More appropriate concurrency control and failure recovery methods used to support long transactions.
Fast Performance	Thousands of sub-components are retrieved and displayed in seconds.	A single view requires multiple joins and many individual accesses.	Designed to retrieve large amounts of data at once.

Table 1.1. DBMS Support of Engineering Design Tool Characteristics (11)

1.2 *Problem Statement*

Currently, Ada can interface with RDBMS, such as the Ada embedded statements of the ORACLE RDBMS (Ada/OCI). RDBMS that provides the ability to model information organized as records has widely been implemented in Management Information Systems (MIS) applications. However, the RDBMS has its limitations for more complex applications, such as computer-aided engineering design. Object-oriented concepts have been implemented increasingly in the field of software engineering. OODBMS, one of implementations using an object-oriented model, includes most of the benefits of RDBMS. In addition, it has the ability to manipulate complex, heterogeneous data. Current OODBMS such as ObjectStore are written in C or C++. These OODBMSs are tightly bound with C or C++. To extend Ada's capability in area of complex, data-intensive applications, there must be a way that Ada can interface with OODBMS.

1.3 *Research Objectives*

The primary purpose of this thesis is to demonstrate that an Ada binding to ObjectStore can be accomplished, and that the interface functionality is similar to the existing C/C++ binding to ObjectStore. Furthermore, the performance of the Ada interface should approach the performance of the C/C++ interface.

Ada is a very powerful programming language and was designed for problem domains needing a software-intensive system. It requires efficiency, reliability, and maintainability. An Ada interface to ObjectStore should not lose these good abilities. Also this thesis should identify any possible limitations related to the interface.

1.4 *Approach*

This thesis effort resulted in the development of an ObjectStore binding for Ada. ObjectStore is written for C/C++. The basic approach employed in this effort consisted of the following:

- Created parallel data types of Ada and ObjectStore. In order to safely convert data that cross the interface, parallel data type needed to be created first. Ada supports scalar, composite, access, private and subtypes, and derived types. A type declared in Ada should have a parallel data type implemented in ObjectStore.
- Implemented interface functions. A series of predefined pragma `INTERFACE` and pragma `INTERFACE_NAME`s were established to link to the ObjectStore function's. Several interface packages were implemented according to the classification of the ObjectStore functions.

- Implemented test programs using implemented interface packages. The test program verified the functionality of the new implemented interface functions. Before a test program was compiled, a database schema was implemented. Binding to the ObjectStore library was accomplished at compile time by the Ada linking facility.

1.5 Materials and Equipment

This research effort utilizes ObjectStore, version 1.2 and development facilities on a Sun Sparc II workstation. Verdix Ada, version 6.0, is used to compile Ada programs and the Ada to ObjectStore interface programs.

1.6 Document Summary

Chapter 2 describes properties of ObjectStore and describes ObjectStore's abilities in support of computer-aided design applications. Furthermore, this chapter describes data persistence, and how a programming language that can extend its capability of handling persistent data through a database. This chapter also describes previous research in interface programming from Ada. Chapter 3 summarizes the prototype of Ada/ObjectStore designed by Object Design, Inc. (18), and present a design of Ada/ObjectStore via the ObjectStore C library. A comparison of Ada/ObjectStore and C/ObjectStore performance after Ada has extended the ability of data persistence is described in chapter 4. This chapter also discusses some problems encountered in the effort of implementating the Ada/ObjectStore interface. Chapter 5 includes conclusions reached regarding the objectives of this thesis and recommendations for further research.

II. Literature Review

2.1 Overview

In order to begin developing a set of interface programs written in Ada that provide access to ObjectStore, an OODBMS product written in C/C++, we need to know some concepts and techniques related with this topic. One key area is to compare intercommunication characteristics of Ada and C/C++. To achieve this requirement, a review of some features of Ada and C/C++ was conducted. These features are abstract data type, data persistence, and current examples of Ada bindings, including X windows and ORACLE.

2.2 Overview of ObjectStore

ObjectStore, developed by Object Design, Inc., is an object-oriented database management system (OODBMS). It provides a tightly integrated language interface with the features of data management found in traditional DBMS. ObjectStore was designed to provide a unified programming interface for both persistent and transient data.

OODBMS contain capabilities of data management as with traditional DBMS. In addition, OODBMS more directly integrates with an object-oriented programming environment. Therefore, the advantage of the OODBMS over the traditional DBMS is that it provides both data persistence and expressibility (19). In a traditional DBMS, transient data are stored in variables in the programming language, and persistent data are stored in the database. Programmers explicitly convert data between transient and persistent states. However, object instances in an OODBMS application are either persistent or transient. The persistent data in ObjectStore is provided by overloading the C/C++ 's memory allocation operator. Transient data is provided with the ordinary operators of C/C++. ObjectStore provides not only persistent and transient data, but, for more efficient data handling, it provides the developer with a single view of memory by dividing the memory space into program memory and database memory. Persistent data stored in ObjectStore are handled by C/C++ programs exactly the same way as transient (non-persistent) data are (17).

ObjectStore is an object oriented database management system. It provides the data query and management capabilities of a traditional database. In addition to the capabilities of a traditional database, it provides the flexibility and power of the C++ object-oriented programming language and the versioning mechanism to support creation and manipulation of alternative object versions. The versioning mechanism enhances parallel work on shared data (17). To group objects together, ObjectStore provides collections which provide a convenient means of storing and manipulating objects. This feature is not supported by C++ and most DBMSs (14). Collections

are abstract structures which resemble arrays in traditional programming languages or table in relational DBMS. ObjectStore collections provide a variety of behaviors, including ordered or unordered collections (lists), and collections that either do or don't allow duplicates (bags or sets). These are commonly used to model one-to-many and many-to-many relationships. They also provide a domain for iteration and for the execution of queries (17).

ObjectStore's unique Virtual Memory Mapping Architecture (VMMA) achieves its performance by using memory mapping, caching, and clustering techniques to optimize data access. The key features of ObjectStore's virtual memory mapping architecture allows persistent data to be handled exactly the same way as transient data, minimizing overhead of retrieving and manipulating large amounts of data, and managing versioned data in a way that does not slow access to non-versioned data. In addition, ObjectStore performs effective associative access and optimizing of queries. These techniques formulate efficient retrieval strategies and minimize the number of objects examined in response to a query (17).

ObjectStore applications require three auxiliary processes for their execution: the ObjectStore Server, the Directory Manager, and the Cache Manager. These processes are started automatically when an ObjectStore application starts. Most users never have to worry about starting or stopping them. The Server handles all storage and retrieval of persistent data. The Directory Manager manages a hierarchy of ObjectStore directories by storing its information in a directory database. The Cache Manager manages an application's data mapped or waiting to be mapped into virtual memory (17).

There are four approaches to using ObjectStore: (17)

1. the C library interface,
2. the C++ library interface without class templates,
3. the C++ library interface with class templates, and
4. the C++ library interface with class templates and the ObjectStore DML.

The C++ library interfaces involved in application systems depend on the compiler used. For the C++ library interface without class templates, the compiler used is based on AT&T's cfront. For the C++ library interface with class templates, the C++ compiler used should include the ANSI Draft Standard. The Object Design C++ compiler supports class templates and ObjectStore's DML, which provides clarity and convenience to access database. Furthermore, the Object Design C++ compiler allows applications that mix these approaches freely; a program could perform some queries using the DML and some queries using the C++ library interface.

2.3 *Programming Language and DBMS*

For a long time, programming language designers have tried to find out an effective way of handling long term storage. Data, if required to survive a program activation, needs to be stored in a file or a DBMS. However, the data structures in traditional DBMS and in programming languages are very different. The traditional DBMS only supports limited data types, but most programming languages support complete data type systems. Now, object-oriented design (OOD) is a new way of thinking about problems using models that are organized around real-world concepts. Currently, the OOD methodology has been implemented in programming languages (OOPL) and database systems (OODBMS). Because they provide the same methodology to handle objects, they provide an advantage for data persistence. The following describes the characteristics of data persistence, defines persistent programming languages, and why we need programming languages that interface to a DBMS.

2.3.1 Data persistence. Persistence is the ability of the programmers to have their data survive the execution of a process in order to eventually reuse it in another process. Persistence should be orthogonal to type. The user should not have to explicitly copy data to make it persistent.

Booch (7) defines persistence as follows:

> Persistence is the property of an object through which its existence transients time (i.e. the object continues to exist after its creator ceases to exist) and/or space (i.e. the object's location moves from the address space in which it was created).

Persistent data should have the following properties (2):

1. Persistence independence: the persistence of a data object is independent of how the program manipulates that data object. Conversely, a fragment of program is expressed independently of the persistence of data it manipulated. For example, it should be possible to call a procedure with either persistent or transient objects as parameters.
2. Persistence data type orthogonality: persistence should be a property of arbitrary values and not limited to certain types. All values should have the same rights to persistence.
3. Persistence transparency: persistence is transparent when the programmer is not aware of how the data maps between memory and storage.

Harper, in "Modules and Persistence in Standard ML" (10:26-27), pointed out that the most general notion of persistence, called object persistence, consists of viewing all objects as existing in persistent storage, with transient storage serving only as a cache for quick access. Each object is

identified by a persistent identifier, or PID, which is the address of that object in persistent storage. The heap is garbage collected as usual so that only accessible objects are preserved. The garbage collector is for efficiency, which depends on what algorithm is used. In order to ensure that all accesses to persistent data are type safe, each object must have its type associated with it. Some sort of run-time type checking must be involved. A persistent environment must associate the type of a structure with the object in persistent storage.

Cardelli (8:37-39) classified persistence strategies and sketched three different persistence models. These correspond to three different semantics for intern-extern. Extern is defined as the operation that copy objects from transient to persistent memory, and Intern is a symmetrical operation for those objects. These strategies are the fetch-store, load-dump, and lock-commit model. Cockshott (3:236) gave a similar but more detailed view of addressing mechanisms for persistent objects. The following is a summary of some of the categories:

- The Fetch-Store Model

  This model is backup storage for transient objects. The association between internal and external objects is mediated by handles. Extern makes a copy of a transient object in persistent storage, associating it with a handle. Many calls to extern on the same object and different handles will make many independent copies. Calls of extern on two objects which share a substructure will duplicate the substructure. Intern has the same functions but opposite direction, copying objects from persistent storage to transient storage. Sharing is only preserved within persistent objects.

- Core Dumping, or session persistent

  A simple way of providing persistence is to make PIDs machine addresses and dump the whole core at the end of a session and reload it at the start of the next session. This is a simple technique and this gives us very efficient use of disk storage, as data is held in contiguous storage. Garbage collection and space recovery is simple too. However, the shortcoming is that it will not be able to hold a collection of data that is larger than its RAM since they assume that the whole collection of data is loaded into RAM at the start of each session. Another considerable cost is the time required to startup and close down. The user must wait for the whole image to swap in or out at the start or finish of the session.

- Use of Virtual Memory, paged or segmented.

  This technique is to make the PIDs virtual addresses in a paged or segmented store. It is not necessary to dump or reload the entire image. Instead, it can be done incrementally, a page or segment at a time as needed. Implementations of virtual memory are transparent to users.

It allows multiple users running programs concurrently. Each program is given the illusion that it is using physical memory alone. However, it will work with degraded performance.

- Multiple Address Space Models

  In this implementation, the PID is charged according to which address space the object contained is currently residing in. If it is resident in RAM, the PIDs are converted to RAM addresses. If it is on disk, the PIDs are represented as disk addresses. A search is made of a memory resident table called the PIDLAM. This table holds a two way mapping between PIDs and local addresses. These are disk addresses and RAM addresses respectively. Every object that was brought in from disk in this session must have an entry in the PIDLAM. The drawback of this technique, however, is that a complex body of software is needed to manage the PIDLAM. Any algorithm used is not just for simplicity. The current PS-Algol uses this technique.

- Associative PID Addressing & Paged Virtual Memory

  This puts another level of addressing above virtual memory. The PIDs here are names of objects rather than addresses. A combination of associative memory hardware and firmware maps these names onto paged virtual memory.

2.3.1.1 Persistent Programming Languages. A persistent programming language is a programming language that provides the ability of data persistence. There are several approaches to providing persistent data services: files, special hardware devices, and databases (19). Most programming languages do not provide this ability. Atkinson pointed out that in any program written with a non-persistent programming language, there is usually a considerable amount of code, typically 30% of the total, concerned with transferring data to and from files or a DBMS (2). Much space and time is taken up by code to perform translations between the program's data and the form used for the long term storage medium. Therefore, the main advantage of persistent programming languages is quite clear. That is, it favorably affects program length, program development time and program maintainability. To discuss persistent programming languages, we need to look at what languages should provide and what abilities are required for persistence.

- Data type completeness:

  The basic requirements for data persistence are persistence independence and data type orthogonality. However, a complete type system should have a method that stores persistent types, such as a schema generator that generates a schema in a DBMS. More specifically, data persistence in programming languages is achieved through data type persistence. The persistent data type works with a data type checking algorithm (data type checking will be

mentioned later) to protect data across the boundary between storages and application programs. Both languages, Ada and C, provide base data types and abstract data types, except inheritance. Both provide data type completeness, and, in some cases, Ada is better than C in data abstraction. Unfortunately, as with most programming languages, they do not provide any way to store data types.

- Memory allocation and deallocation:

  Ada and C provide three kinds of memory allocation: static (global), automatic (stack), and dynamic (heap). Static and automatic memory normally is allocated at block entry. Dynamic memory is explicitly allocated memory. In Ada memory deallocation works automatically. However, in C deallocation is done explicitly. Garbage collection, or memory deallocation, is a factor of performance for languages.

- Type checking:

  It is dangerous to allow languages without a strong type checking mechanism to handle persistent data. Languages must provide type checking mechanism to protect against a system crash during run time. Because many objects are transferred between the disk and memory while the system is running, any type mismatch will cause the system to exit abnormally, or even worse, cause some erroneous data to be stored. Languages should provide a method of storing and retrieving persistent data as well as a description of its type and a method of type checking. Type checking is another weak point of C. Ada provides strong type checking to prevent run time errors. Moreover, Ada's exception facility provides a more elaborate way to handle errors at run time. C does not have those benefits.

- Persistence through reachability or declaration:

  Persistent programming languages provide the ability of data persistence through reachability or declaration. Programmers need to understand how data persistence is provided by the language they use.

  - Persistence through reachability:

    This approach has one or more persistent database roots and makes every object that is reachable from these persistent. This was the approach used in one of the earliest persistent programming languages, PS-ALGOL (2)

  - Persistent through declaration:

    This approach is to declare data structures that are persistent. For example, to declare a structure PERSON that is persistent, all objects created for PERSON are persistent. Languages may provide a different operation for allocating persistent or transient ob-

jects, like ObjectStore does (17). Besides the fact that objects can be declared to be persistent, classes can be declared to be persistent, too. Classic-Ada provides persistence by declaring classes to be persistent (22). However, the persistence ability in Classic-Ada has limitations; all objects under the class declared persistent must be persistent.

- Memory management:

  Memory is used to temporarily store a program and its data. Programming languages provide a uniform memory system. That is data is uniformly distributed in the memory regardless of its properties. All addresses of pointers are memory addresses. However, for persistent data, PIDs representing addresses of objects on disk are required. Also new operators are required to enable them to dereference PIDs. Languages, for example PS-Algol (3:243) use multiple address spaces, separate the memory so it is persistent and transient. All persistent objects are stored in the side of persistent memory, and all transient objects are stored in the side of transient memory. Other languages such as LISP and PROLOG (23) are implemented in a different way: Persistent Memory. A persistent memory system that is based on uniform memory abstraction eliminates the distinction between the computational (transient) and long-term storages (persistent). The uniform memory abstraction is that a processor views memory as a set of variable-sized blocks or objects interconnected by pointers. All objects that are in the transitive closure of the persistent root are persistent, and vice versa for transient objects.

2.3.2 Programming Language Interface to DBMSs. Ada, a procedural programming language, is based on constructs such as loops, branches, and if/then pairs. These programming languages provide good performance in computations, but in the case of intensive interrelated data retrieval and manipulate, they provide data store and query facilities which are far behind those of a DBMS. A DBMS provides concurrency control, and failure recovery for data that it stores. Also a DBMS supports a transaction mechanism to ensure that the persistent value including updates produced by transactions are executed to completion. However, programming languages alone do not provide recovery algorithms that acts on persistent values. Persistent programming languages have gained the advantage of simplicity and maintainability (2). But, to obtain the best advantage, allowing a program language access to a database is an advantageous approach.

2.3.3 Approaches to the interface. A programming language interface to a DBMS can be accomplished by two methods–loosely coupled or tightly coupled. The following discusses the two methods and their trade offs.

2.3.3.1 Loosely Coupled - ORACLE and Ada. ORACLE is a Relational Database Management System (RDBMS) and can be accessed and manipulated by an application program written in Ada. ORACLE provides a set of host language calls that can be included in application programs. An Ada program that embeds these calls is known as an Ada/OCI program (16).

Ada/OCI provides a direct interface to the ORACLE RDBMS. The SQL of ORACLE is a non-procedural language. That is, most statements are executed independently of the preceding or following statement (16). Ada is a procedural language and it has limitations on data management. However, under Ada/OCI construction, programmers can write software that combines the advantage of SQL and Ada. The basic structure consists of several statements. For example, a program establishes communication with the ORACLE RDBMS by issuing the `LOGON` call. Communication takes place via the `Logon Data Area` that is defined within the user program, and the `EXECUTE_SQL` call executes a specified SQL statement.

ORACLE fetches and stores data objects into and out of the user program by directly accessing the data via its actual address. Because of this requirement, if the data in Ada accomplish this accessing, all scalar objects are represented as record types with a single component of the scalar type. For example, the type used to represent short integer (16-bit) to ORACLE for database operations is defined as follows (16):

```
type oracle_short_integer is record
        int : short_integer;
end record;
```

ORACLE performs data conversions for data types provided by the user program. On retrieval operations, ORACLE converts from the internal format of the data as stored in the database to an external format as defined by the user program. On storage operations, ORACLE converts from external to internal data types. ORACLE may store characters in ASCII strings and numbers in a variable length scaled integer format.

The disadvantage of OCI/Ada is they are loosely coupled. Rumbaugh pointed out (19) that this scenario is unattractive and the fundamental problem is twofold. The problem of this implementation is that they are totally different languages. The interface is through a set of language calls which are implemented by ORACLE. Moreover, ORACLE does not provide the complete capability of data persistence. The programmer must explicitly convert data between persistent and transient formats. This conversion causes inconvenience for application developers.

2.3.3.2 Tightly Coupled - ObjectStore and C/C++. Section 2.2 shows ObjectStore is a tightly integrated language interface for the features of data management. ObjectStore was

designed to provide a unified programming interface to both persistent and transient data. Data that are allocated in persistent memory with an overloaded C++ new operator are persistent. Otherwise, they are transient. Programmers handle the persistent and transient data with no difference. Because the capability of complete data persistence is achieved, the explicit I/O and data conversion are not required. Furthermore, ObjectStore provides some advantageous abilities for manipulating data. For example, the collection provides the ability to handle aggregate data structure, and it allows application programs to be developed more simply and readily maintainable.

2.4 Ada and C/C++ Communication

Ada, from several points of view, provides data abstraction. Ada's package can define a set of values or data structures and a set of operations that manipulate the data structure it defines. A package consists of two parts: package specification and body. The *specification* contains the declarations of types, objects, and subprograms and acts as an interface between the package and client programs. The package *body* contains the actual code for the subprograms declared in the specification. Data declared in the specification is accessible from the external world, but data contained in the body is hidden from the outside. However, a current shortage in Ada's data abstraction is that it doesn't support inheritance.

Although Ada is not truly an OOPL, Ada does support some of the major concepts of the object-oriented philosophy in the area of data abstraction, namely overloading, encapsulation (packages), information hiding (private types and package bodies). These features make Ada a quite suitable OOPL. The following discusses those features in Ada and compare those features to C.

2.4.1 Information Hiding. The information-hiding feature of abstract data typing means that objects have a "public" interface. However, the representations and implementations of these interfaces are "private". The abstraction mechanism that enforces the access and update of objects with user-defined types is encapsulated. Hence, it can only be performed through the interface operations defined for the particular type. Ada provides information hiding. For example, a data structure, stack, is defined to "private". The type name of stack will be allowed to export from a package, but its internal structure is invisible to the user program. Also Ada provides a greater degree of information hiding or encapsulation. Fox example, the stack can be completely concealed in the package body. Because of encapsulation, only one stack is required. The programs will be simplified when the encapsulated stack is used. C does not support information hiding and encapsulation.

2.4.2 Overloading. Overloading allows operations with the same name but different semantics and implementations, to be invoked for objects of different types. This is one of the most powerful and useful concepts of object orientation. The common examples are overloading operators and overloading names. In almost all languages, the arithmetic operators "+", "-" and "*" are used to add, subtract, or multiply integers or floating-point numbers. These operators work even though the underlying machine implementations of integer and floating-point arithmetic are quite different. The compiler generates object code to invoke the appropriate implementation based on the kind of the operands. Ada and C support overloading operators, but only Ada supports overloading names. Names, in any language, are used to denote entities. Ada is good for large scale systems in which the name space may contain more than hundreds of names. In order to avoid problems using names already defined, Ada allows the overloading of certain names. This facility is specially useful for subprograms and enumeration literals. One exception is that object names cannot be overloaded. One example of overloading a subprogram's name, CLEAR, is as follows:

```
procedure CLEAR (THE_VALUES : in out VALUES);
procedure CLEAR (THE_MATRIX : in out MATRIX);
```

2.4.3 Polymorphism. Polymorphism generally represents the quality or state of being able to assume different forms. When applied to programming languages, it indicates that the same language construct can assume different types or manipulate objects of different types. Fairbrairn (3:70) pointed out that with polymorphism the function works just as well whether the type is, for example, **int** or **char**. The idea is to replace the irrelevant details with a type-variable that can be filled in later. Overloading is analogous to polymorphism, such as "+", an overloading operator, can apply to different types of objects in which base types are INTEGER or FLOAT. To maximize the re-usage of software, it is important to be able to parameterize software components so that the same blueprint can be used in type-safe fashion for different applications. Ada's generics support parametric polymorphism. In contrast, C has no parametric facilities. The common practice is to use the C preprocessor (a macro expander) to duplicate text with suitable replacement in order to simulate generic instantiation (21). This mechanism is purely lexical: there are no syntax or semantic checks attached to the macro definition nor to each of its expansions. Another way is using "void*" as a parameter. Because the object is typed as "void*", a cast is necessary when using it. The weak type checking of C makes programmers responsible for types matching in application programs, and this will cause the most common type of run-time errors.

2.5 *Interface Programming from Verdix Ada*

The main issue of binding between Ada and C is to match parameters. Objects that can be matched in these two languages are based on their type systems. Fortunately, types in Ada and C can be manipulated to match. C functions act like functions or procedures in Ada depending on whether or not they return a value. The following are basic concepts to accomplish the interface.

2.5.1 Create Parallel Data Types. The VADS Programmer's Guide points out that the first step in creating an Ada interface to a subprogram in C is to "create parallel data types" (24). The parallel type, or data structure does not mean the type's name are identical in Ada and C, but the composition, length, and alignment of the component of that type are required to be identical. From the programmer's guide of Verdix Ada, two basic approaches are available for creating parallel data types:

1. using parallel data types known by the programmer from reading the vendor's documentation, and
2. using Ada representation specifications.

Ada representation clauses allow the Ada programmer to define an exact duplicate of the physical layout of any data type in another language once it is known. For example, the type `INTEGER` in Ada corresponds to the type `int` in C; the type `SHORT.INTEGER` is equivalent to the the type `short` in C. Both `int` and `short` represent a 16 bit integer. Table 2.1 shows some base types that parallel between Verdix Ada and traditional C. Ada allows type specifications that are largely independent of the implementation. For example, the type `SHORT_INTEGER` in Ada can be defined to equal the type `unsigned short` in C. Type, storage, record layout, and alignment can all be controlled. When the underlying representation of a type has no analogue in Ada's language, the data type can be defined by the programmer using Ada representation specifications and UNCHECKED_CONVERSIONs. For example, the type `char` is used both to represent a character or a byte integer value. There is no exact Ada analogue to this behavior, but the generic function `UNCHECKED_CONVERSION` offers a method for controlled easing of type conversions: Ada's `TINY_INTEGER` can be used for numeric representations and type `CHARACTER` can represent a character value (24). One thing that is important when using the parallel data types is that the `PRAGMA INTERFACE` permits only 32-bit or 64-bit scalar values to be passed. Consequently, when you pass `INTEGER` you can pass it by parallel `INTEGER` variables in C. But, when you pass `SHORT_INTEGER` variables to a C function, you must pass them by address. System address is a predefined attribute in Ada. The value of this attribute is defined in the package `SYSTEM`.

Traditional C	Verdix Ada
int	INTEGER
long	INTEGER
short	SHORT_INTEGER
char	CHARACTER TINY_INTEGER
float	SHORT_FLOAT
double	FLOAT

Table 2.1. The parallel data types between Verdix Ada and traditional C

In compound types, such as ARRAY or RECORD, the same approach can be taken. Both C and Ada associate the label of compound types with a base address and offsets provide access to individual components of these types. For the Ada programmer, as long as the compound types are composed of equivalent simple data types, the offsets will be calculated similarly and the structure of compound types will be identical. Verdix recommends that the SYSTEM.ADDRESS of the first element of an Ada array be sent to pass the array to C. For record types, the pointer which contains the ADDRESS of the first element is the best way to send the record to C. Pointers and address types are implementation specified. Ada's host conventions usually allows the use of Ada pointer and address types parallel to their C counterparts. If for some reason, host conventions are not followed, representation specifications can be used to fit the size and range of the data type (24).

String types in Ada and C are different. A C string is simply a pointer that points to the first character. The string is terminated by a null character. In another words, there is no explicit length for C to store. In Ada, however, a string is represented by a pointer to an unconstrained array of characters and it needs an explicit length provided by its attribute, LENGTH. An Ada subprogram that calls a C function passing a string as a parameter should be prepared to make the necessary conversions.

2.5.2 Declare External Subprograms. After parallel types have been designed, interface packages need to be implemented to access programs written in C. Ada provides an ability of interface to other languages. A subprogram written in C can be called from an Ada program provided that all communication is achieved via parameters and function results (1). This is accomplished by a predefined PRAGMA INTERFACE_NAME to establish a link from the Ada procedure or function name to the corresponding procedure or function written in C The Verdix PRAGMA INTERFACE allows Ada programs to call subroutines defined in C with:

```
pragma INTERFACE_NAME (Ada_subprogram_name, subprogram_link_name);
```

Encoding Scheme	Types	Encoded Symbols
Basic Types	void	v
	char	c
	short	s
	int	i
	long	l
	float	f
	double	d
	long double	r
	...	e
Type Modifiers	Unsigned	U
	const	C
	volatile	V
	signed	S
Standard Modifiers	pointer *	P
	reference &	R
	array	[10]A10_
	function	F
	ptr to member	S::*M1S

Table 2.2. The function name and parameter encoding scheme

The subprogram_link_name argument may be formed from a string literal, a constant string object, or a catenation of these operands. C, unlike Ada, is case sensitive, so subprogram_link_name in pragma INTERFACE must be same as the case of the function written in C. A pragma is allowed at the place of declaration, and must apply after Ada_subprogram_name used in its pragma INTERFACE_NAME has been declared. The Ada compiler handles parameter pushing and target language compiler naming conventions and checks to make sure the parameters are allowed in the target language.

2.5.3 Accessing C++ and ObjectStore's Extended Functions. C++ is an extension of the C language, implemented not only to add object-oriented capabilities but also to redress some of the weaknesses of the C language. Many features are added such as, inline expansion of subroutines, overloading of functions, and function prototypes (19). It was originally implemented as a preprocessor that translates C++ into standard C. After the functions of C++ are translated to C, Ada can access the intermediate C functions as described above. In "Type-safe Linkage for C++" (4) an encoding scheme for functions written in C++ that can be linked by C is presented. The encoding scheme is designed so that it is easy to determine, if a name is an encoded name. What name the user wrote, what class (if any) the function is a member of, and what the types of arguments are in the function. The types are encoded as in Table 2.2.

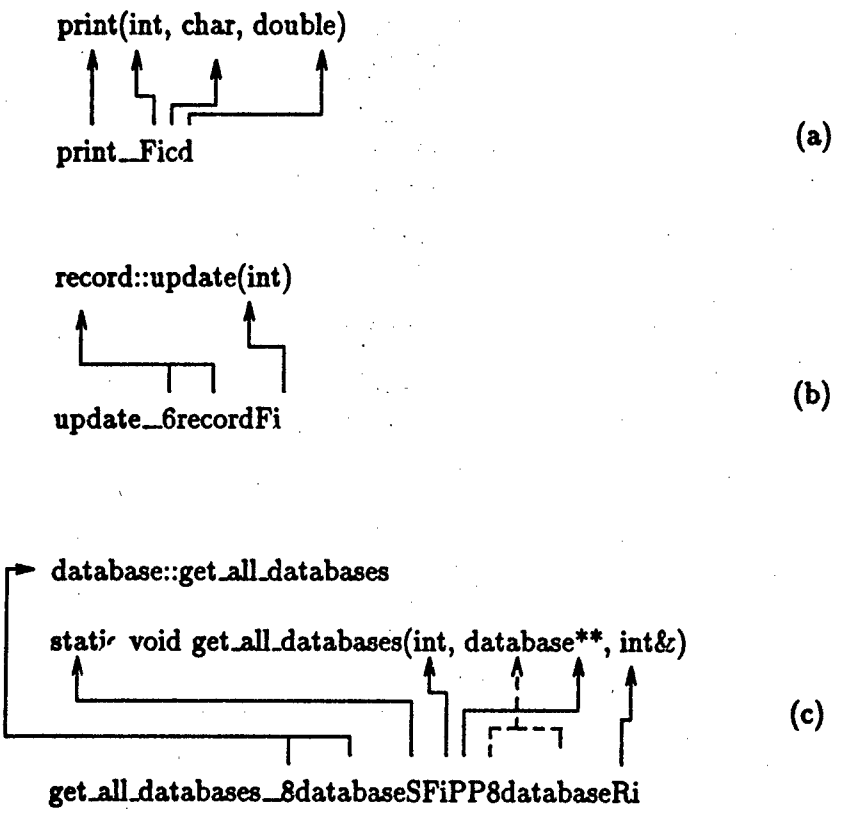
For a global function, the name is encoded by appending _F followed by the signature. Figure 2.1(a) shows an example. For a member function in a class, first, the class name that contains this member function is appended to a number which represents the length of the class name. The encoded class name then is appended by the member function name and two underlines. The design decision to involve a length is to avoid terminators. Both the class name and user defined type name require their length in the encoding scheme. Figure 2.1(b) shows how to encode record::update(int).

The ObjectStore function are exactly C++'s syntax, so it should be the same as C++. This can be encoded and then access provided to those encoded functions from Ada. Rosenberg (18) designed a prototype of Ada/ObjectStore, which is an interface that allows applications written in Ada to access ObjectStore. The interface of Ada/ObjectStore is actually done by accessing C++ encoded (mangled) names which are kept in ObjectStore's library. One example, database::get_all_databases, is shown in the Figure 2.1(c). The length in here shows that it can be used to represent the length of a user defined type name.

2.5.4 Ada binding to X window. Ada binding to X windows is a good example for Ada programs accessing C functions. The X window system, developed in the mid 1980s, changed the way that user interfaces were developed. The X window system, or X, is a high performance, device independent, network transparent window system that allows for the development of portable graphical user interfaces (20). X windows manages what is seen on the display screen. The programmer is not constrained by any particular policy. As a result, X provides mechanism rather than policy (12). But the X window system was implemented in the C language. Therefore, there was no way for Ada to access X windows. Recognizing the benefits of the X window system, some members of Ada community began working on ways to access the X window system from Ada. One successful method is by way of a binding.

Under a Software Technology for Adaptable System foundation contract, in 1987 the Science Applications International Corporation (SAIC) developed Ada bindings to Xlib, which is written in C. The actual Ada interface is accomplished through the use of Ada `pragma interface` statements. A `pragma` conveys information to the compiler. The name of `interface` after `pragma` means the Ada compiler allows subprograms written in another language (7). The configuration of the SAIC binding shows in Figure 2.2 (13).

Some functions are missing from the SAIC binding because of the shortcomings of interface. One shortcoming is the procedure variables required as parameters to function calls (23). A few Xlib functions require procedure variables as parameters to function calls. Ada does not directly



(a) A global function.
(b) A member function in a class.
(c) A ObjectStore member function in the class of database and a ObjectStore defined type, database.

Figure 2.1. Examples for the function name encoding scheme

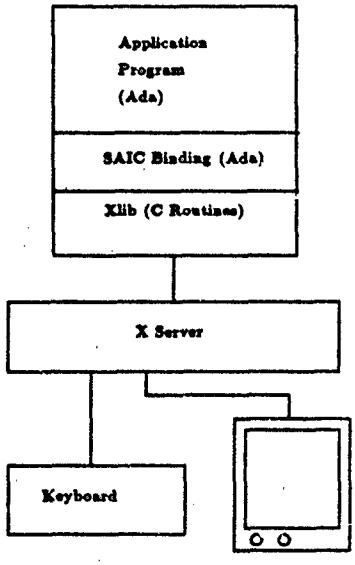


Figure 2.2. Application Program Configuration Using the SAIC Binding

support procedure variables. Another one is the representation of event types as enumerated types. In the C version the events are represented as integers with a large block of consecutive integers, beginning with zero, reserved by the X Consortium for future use. X was designed to be easily extensible. But, by using enumerated types for event types, adding new events is nontrivial because the programmer needs to ensure the position in the enumerated type declaration matches the event numbers used in the C code. Enumeration types for events limit the extensibility of Ada/X (23).

2.6 Summary

OODBMS is a new generation database system. Because traditional DBMSs have proven inadequate in some applications, the OODBMS is designed to widen the applications of database technology. The ObjectStore is currently one of the commercial OODBMSs available. It combines the paradigms of object oriented programming language, C/C++, and capabilities of DBMS. Ada bindings to C have been implemented in some applications. One milestone is Ada bindings to X windows. For Ada bindings to a database, the important factor in this effort is how Ada deals with persistent data. Some features of persistent data, type systems in different languages have been examined. The purpose of this thesis is to design an Ada binding to ObjectStore to extend Ada's capabilities in the area of OODBMS.

III. Design and Implementation

3.1 Overview

To design of an interface between Ada and ObjectStore the designers require a good understanding of the two languages and their capabilities as related to the interface. Based on these requirements chapter 2 described programming language interface to a DBMS, Ada and C/C++ communication, and interface programming from Verdix Ada. In this chapter, the design is then compared to different models and the best model is chosen to accomplish the task. Finally, the implementation follows the design paradigm to approach functional completeness. For performance measurement, the code is instrumented with timing commands where appropriate. Testing is then accomplished to verify functionality as compared to the ObjectStore.

3.2 The Prototype of Ada/ObjectStore

The prototype of Ada/ObjectStore implemented by Object Design, Inc. is a high level design providing to basic interface facilities to Ada. The Ada/ObjectStore interface should be complete and transparent. Performance is an another important factor for evaluating the interface and it should be as close as C/C++ accessing ObjectStore. Completeness requires that all functionality in ObjectStore should ideally be accessible from Ada. Transparency should be provided so that an Ada programmer would not need any knowledge of C or ObjectStore's native facilities in using the Ada/ObjectStore interface. Finally, we desire the performance of the Ada/ObjectStore interface to be as fast as the performance of the ObjectStore C/C++ interface. That is, it provides near virtual memory access speed to persistent Ada instances. The following is a summary of Ada/ObjectStore(18).

3.2.1 Ada/ObjectStore interface. Object Design Inc. provided a prototype interface between ObjectStore and Ada. This interface is based on the interface facilities supported by Verdix-Ada. These are `PRAGMA INTERFACE`, `PRAGMA INTERFACE_NAME`, `PRAGMA LINK_WITH`, and `PRAGMA INLINE`. Object Design considered the following options in designing the interface (18):

1. Object access

   This would provide Ada abstract types for objects actually represented and accessed in C/C++. Each object in Ada would have one counterpart in C/C++ stored in the ObjectStore database. The Ada type would have no Ada level functionality at all, but would uniquely identify a persistent C/C++ object. All functions affecting persistent objects would be written in C/C++, not in Ada, using the existing interface (see Figure 3.1). Ada sim-

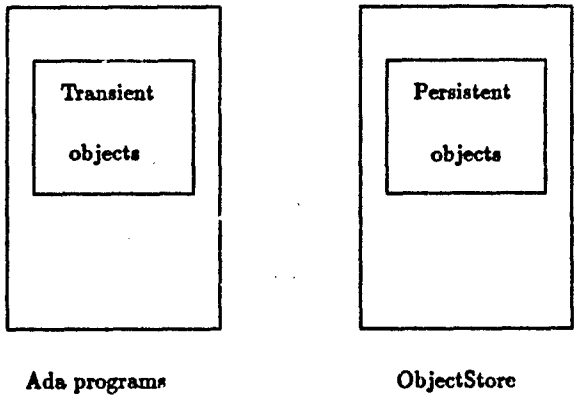


Figure 3.1. Object access

ply handles the OID of associated ObjectStore objects via the interface from C/C++. The disadvantage of this simple interface is that this would not provide transparency for Ada programmers. ObjectStore has a characteristic that treats transient and persistent data in the same way. In this option, Ada programmers lose this characteristic because when they want to deal with persistent data they must send data to C/C++ to store in a database. Furthermore, this option implies a rigorously separating persistence store for objects from the Ada code and data space. Therefore, this option needs a large number of foreign function calls between Ada and C/C++. These calls will affect performance and may prohibit applications.

2. Basic persistent Ada instances (Manual schema generation)

   This approach directly represents Ada objects in the ObjectStore database. In this approach, Ada programmers have the same capabilities as ObjectStore programmers in manipulating both transient and persistent data (see Figure 3.2). The basic requirement of this interface is to provide a set of Ada declarations for the kernel functions of the ObjectStore library interface, which has functionalities of ObjectStore dealing with persistent data. In this approach, transparency to Ada programmers is achieved. The Ada programmers may apply these kernel functions without knowledge of the C language. However, because of the lack of a preprocessor to generate the schema transferring data types from Ada to ObjectStore, they will have to separately specify C descriptions of the Ada types. A C/C++ macro facility can be provided to ease these definitions.

3. Advanced persistent Ada instances (Automatic schema generation)

   This option gives total transparency to Ada programmers. The manual schema as described

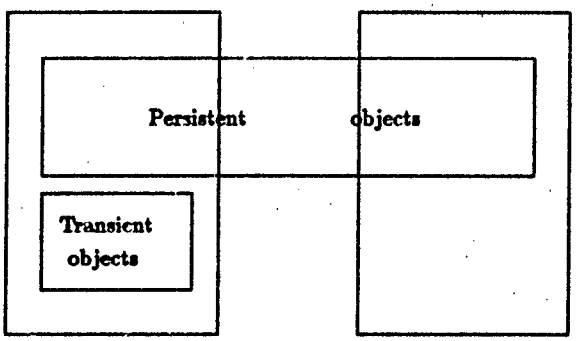


Figure 3.2. Manual schema generation

above needs to separately specify C/C++ descriptions of Ada types in a C/C++ macro language. This option provides a preprocessor to parse Ada type definitions and build ObjectStore compilation schemas directly. This is feasible because the ObjectStore type system is available at run time and is general enough to represent virtually any type.

3.2.2 Compare Ada/ObjectStore and ObjectStore. The Ada/ObjectStore interface provides several of Ada's portable types related to ObjectStore (a list of the most fundamental types is in Table 3.1) and some of the kernel functions. Table 3.2 shows those functions that parallel to ObjectStore's functions in C/C++ library. The functions in Ada/ObjectStore dealing with the database are enough to manipulate persistent data. For efficiency, Ada/ObjectStore provides an interface to C++. An example of the function "DATABASE_CREATE" is as follows:

```
function c_database_create(PATH: ADDRESS;
                           MODE: U_MODE;
                      OVERWRITE: OS_BOOLEAN) return DATABASE;

pragma INTERFACE(C, c_database_create);
pragma INTERFACE_NAME(c_database_create,
                      C_SUBP_PREFIX & "create__8databaseSFPCciT2");


function DATABASE_CREATE(PATH: STRING;
                         MODE: U_MODE := 8#664#;
                    OVERWRITE: BOOLEAN := FALSE) return DATABASE is
begin
       return
c_database_create(c_ada_to_c(PATH(PATH'FIRST)'ADDRESS,
         PATH'LENGTH), MODE, B_TO_OSB(OVERWRITE)); end
```

C/C++	OBJECTSTORE TYPES	ADA TYPES
unsigned char	os_unsigned_int8	UNSIGNED_TINY_INTEGER
signed char	os_signed_int8	TINY_INTEGER
unsigned short	os_unsigned_int16	UNSIGNED_SHORT_INTEGER
short	os_int16	SHORT_INTEGER
unsigned int	os_unsigned_int32	UNSIGNED_INTEGER
int	os_int32	INTEGER
int	os_boolean	BOOLEAN

Table 3.1. Fundamental data types of C/C++, ObjectStore, and Ada

```
DATABASE_CREATE;
pragma INLINE(DATABASE_CREATE);
```

The "create_8databaseSFPCciT2" is a C++ mangled name. The function c_ada_to_c provides the facility of transferring a string in Ada to C++.

ObjectStore's C/C++ library interface allows access to many of ObjectStore's features directly from C/C++ programs. These features include (17):

- databases and segments,
- roots,
- transactions,
- references,
- collections,
- queries, and
- versions.

Ada/ObjectStore provides basic functions of databases, roots, and transactions. Segments, references, collections, queries, and versions are not provided.

Because Verdix Ada has `pragma interface_name` and `interface C` options that allow C functions to be called from Ada, if necessary, it should be not difficult to implement an interface that parallels what ObjectStore has. But, it will have some limitations such as relationships which relies heavily on the syntax and semantics of C++. Some features of ObjectStore's interface are missed in the prototype of Ada/ObjectStore. These features are mentioned as follows:

- Collections:

  A collection is an object that serves to group together other objects. Collections in ObjectStore provide a convenient means of storing and manipulating groups of objects. With this

facility, objects in the same class are transparent to programmers. We can use linked lists in Ada data structures to implement the same functionality as the collections have, but it will be complicated. Collections hide a detailed mechanism that manipulates groups of objects from programmers.

- Version management:

  Version management has very important facilities especially for computer-aided design (CAD) applications. These applications support cooperative work by a number of engineers on the same design. Currently, Ada/ObjectStore does not provide version management facilities. That means Ada/ObjectStore is not currently good in CAD applications, or other applications which need to check out data for extended periods of time. Version management needs two classes of functions, configurations and workspaces.

- Exceptions:

  Ada/ObjectStore provides an exception handling facility to report errors that arise. They apply an interface that maps each predefined ObjectStore exception to an associated Ada exception. They are implemented through a routine that utilizes a hash table to determine the Ada exception associated with the signaled ObjectStore exception. It then calls an Ada routine which raises the exception. Ada/ObjectStore's packages `except.a` and `except_b.a` (see Appendix C.12 and C.13) provide this one-to-one mapping. For example, the parameter ERR in the procedure of OS_ADA_EXCEPTION will return an integer that maps to an exception in ObjectStore. An corresponding exception will be raised in this package, if it happens. However, `except.a` and `except_b.a` only define one exception for the purpose of demonstrating that it is possible to convey an error message to Ada when an error is arisen in ObjectStore. For practical application and functional completeness of Ada/ObjectStore, `except.a` and `except_b.a` are needed to implement all exceptions that ObjectStore has.

3.3 Implementation Issues

Because ObjectStore provides four kinds of interface approaches as described at Section 2.2, which one will be used must be decided first. Then, what types defined in Ada and their counterparts in C need to be considered. Finally, an interface is designed and it contains Ada's subprograms to link to their parallel functions in ObjectStore. The following discusses the decision that was made to use the C library interface, compares the type system in the interface, and then describes how Ada/ObjectStore facilities were implemented.

Ada/ObjectStore Functions	ObjectStore Functions (C++ Library)
DATABASE_ROOT_GET_VALUE	database_root::get_value
DATABASE_ROOT_SET_VALUE	database_root::set_value
PERSISTENT_NEW	void*::operator new
DATABASE_CREATE	database::create
DATABASE_LOOKUP	database::lookup
DATABASE_OPEN	database::open
TRANSACTION_GET_CURRENT	transaction::get_current
TRANSACTION_BEGIN	transaction::begin
TRANSACTION_COMMIT	transaction::commit

Table 3.2. Functions of Ada/ObjectStore and their equivalent functions in ObjectStore C++ library

3.3.1 C Library Interface. Clearly, only two kinds of interfaces, C and C++, are concerned. Using C++ library interface with class template DML or the C++ library interface with class templates will be complex because of the need to encode class names in the mangled C names. Furthermore, the DML can not be used in the interface unless a preprocessor is implemented. To implement the class template and the preprocessor in the interface will be more complicated than the C/C++ library interface. The decision to use the C library interface is based on:

1. The syntax of languages:

   Because both Ada and C are not OOPL, they do not provide the concept of **class**. Also, the **syntax** and object defined in Ada and C are almost same. Most functions in the C library can be exactly replicated in Ada. For example, a function to create a database root designed in Ada "function DATABASE_CREATE_ROOT(DB : DATABASE; NAME : string) return DATABASE_ROOT;" is exactly the same as "database_root * database_create_root(database *db, char *name)" in the C library. The interface in Ada is done by directly putting database_create_root in a statement **pragma INTERFACE_NAME**. However, the same function in C++ is a member function defined in class **database**, which is " database_root * create_root(char *name)". Because, in this case, it is in the class of **database**, the parameter of **database** is not required. This is also part of the distinct syntax in C++, which provides facilities pointing to member functions. This is quite different in comparison to C and Ada.

2. The Complexity of Ada/ObjectStore Interface:

   Because C++ was designed using a preprocessor to convert C++ programs to standard C before they are compiled, all C++ functions, as in Table 3.2, can be represented by their mangled names as shown in Table 3.3. The interface of Ada to C++ library is actually done

ObjectStore Functions C++ Library (mangled name)	ObjectStore Functions C Library
get_value__13database_rootFP5_Pvts	database_root_get_value
set_value__13database_rootFPvP5_Pvts	database_root_set_value
nw__FUiP8databaseP5_PvtsiPv	objectstore_alloc
create__8databaseSFPCciT2	database_create
lookup__8databaseSFPCci	database_lookup
open__8databaseSFPCciT2	database_open
get_current__11transactionSFv	transaction_get_current
begin__11transactionSF21transaction_type_enum	transaction_begin
commit__11transactionSFP11transaction	transaction_commit

Table 3.3. Functions of ObjectStore C++ library (mangled name) and the same functions in C library

by the mangled name, which is put in the statement pragma INTERFACE_NAME. Table 3.3 also shows the same function of each C++ mangled name and its corresponding function in C. It is obvious that the C++ mangled name is more complicated than the name in the C library.

The main concern of this thesis is to implement an interface from Ada to access ObjectStore. That means the goal of this design is to prove that the interface, Ada/ObjectStore, could be done and the performance is not greatly altered. The interface of Ada/ObjectStore can be done by using ObjectStore C and C++ library, but it will increase the complexity if the C++ library is used. So the design decision to interface Ada/ObjectStore was decided using C library functions.

3.3.2 Types in the Interface. Ada/ObjectStore provides persistent objects for Ada. However, types written for the schema must be based on parallel types between Ada and ObjectStore. For this reason, types defined in Ada are matched to C in a manually constructed C macro. To implement an interface, types defined in the interface should be considered first.

Ada is a strongly typed language. That means objects of a given type may take on only those values that are appropriate to the type. In addition, the only operations that may be applied to an object are those that are defined for its type. Ada provides a more advantageous type system than C. Private type ability is an example. For interfacing with another language, representation clauses can be used to specify the mapping between types. Section 2.5.1 has discussed the concept of creating parallel data types between Ada and C. Scalar, composite, and access type have been mentioned. Tables 2.1 and 3.1 show scalar types in C and their counterpart types in Ada.

Now, about implementation issues, because the main purpose of involving ObjectStore is to provide data persistence for Ada, a detailed observation needs to be done in which the types in C

can map types defined in Ada. One fundamental concept is that all types declared in the interface map through their base type that is a predefined physical layout. That means an abstract type defined, for example AGE, in Ada it it base type is short_integer, the interface type in C must be short because the physical layout in both types, short_integer in Ada and short in C, are represented in 16 bits. Nothing needs to be done with abstract type AGE. Another example for using a representative clause is as follows:

```
type TEMPERATURE is range -100 .. 130;
for TEMPERATURE'SIZE use 8*BITS;
```

As a result of this declaration, every object of type TEMPERATURE will occupy no greater than 8 bits of storage. For physical alignment, if the user uses such a specification, a sign char should be used in C because of 8 bits of storage. A representative clause provides efficient feature for Ada. However, representation specifications are implementation dependent; a given compiler must process the semantics of each clause correctly. Otherwise, the clause will have no effect (6:324). The programmer must bear in mind that he must implement the same size (or storage) of type in C to correspond to the size declared using representation specification in Ada. It is complicated and unwise to define a type if its size is not 8, 16, 32, or 64 bits.

A floating type can be assigned and described in terms of attributes called precision and range. The precision describes the number of significant decimal places that a floating value carries. The range describes the limits of the largest and smallest positive floating value that can be represented in a variable of that type. The floating types in Ada include the type float. An implementation may also have predefined types such as short_float, which has less precision than float. Ada provides an explicit mechanism to define a precision floating type. By default, a 32-bit word represents the type short_float, and 64-bit for type float. This corresponds to the type float and double, respectively, in C. The number of decimal digits of significance in C is 6 and 15 digits for float and double, respectively. This is the same as in Ada. The programmer needs to notice that a floating type declared in Ada and its corresponding type in C have the same storage size, 32 or 64 bits, for example. For the significant digits of the floating types, Ada provides an ability to define the number of decimal digits of significance but the number of decimal digits must be set equal or less than the maximum digits supported by the Ada compiler and machine used. C does not provide this ability so that the number of decimal digits that are always given by the maximum digits supported by the C compiler and machine used. Because the Verdix Ada and traditional C compiler provide the same maximum decimal digits and the interface is implemented in the same computer, the floating type defined in Ada will not lose its accuracy when it interface with float and double in C, respectively. One release note from Verdix Ada (24) pointed out that the programmer

need beware of passing floating point parameters from Ada to C using pragma INTERFACE. It does not work correctly in some cases when there are more than six words of parameters.

The fixed point type in Ada provides an absolute accuracy. There is no similar type provided in C. The implementation of the interface for the fixed point type in Ada has the same concern as the floating types–accuracy. The Verdix Ada provides 32-bit storage size for the fixed type and its maximum decimal digits is 3 (24). It should be no problem that define a `float` type in C to receive and store the fixed type from Ada.

For predefined type and subtype one needs to trace its parent type (base type) to implement a corresponding type in C. The range constraint of scalar types in Ada provides a better practice for defining explicitly the bounds of its types. The Ada compiler then chooses the appropriate underlying representation. However, in the interface, the base types are more interesting than each of these constrained types; the Ada programmer may design an abstract data type to map to the real world, but he must put a corresponding type in C to store the value from Ada applications.

The compound type in Ada requires an access type to point to it in the interface so that objects of the compound type crossing the interface boundary is accomplished via the access type. A parallel data structure must declare and strictly demand a physical alignment to the corresponding type in Ada. That means the storage size of each component in C needs to align in Ada.

The enumeration type is like the mutual properties of scalar and compound type. It is constructed in a similar way in Ada and C, using a variable that presents an offset from that enumeration type. If programmers declare an enumeration type in Ada and store it as an `int` in C, there is no difference in using an enumeration type than using a scalar type in the interface. There are two possible implementation methods: one is using Ada's attribute `POS` to convert the enumeration variable to an `integer` before it is sent to C, another one is to directly send to C as an enumeration type. Both ways work fine. However, the disadvantage for the former one is that the programmer needs to do conversion task, and an extra integer variable is required. The disadvantage for the latter is that the enumeration type must be known in the interface package, by declaration or by using the `with` clause. Moreover, the Ada compiler will generate a warning message, as follows, if the latter one is used.

```
warning: RM 13.9(4): discrete type arguments to 'C' must be 32 bits wide.
```

Boolean in Ada is a predefined enumeration type. To implement a corresponding type in C is the same as the enumeration type described above.

Table 3.4 summarizes all types in Ada and their counterpart types in C.

Size	ADA TYPES (Verdix Ada)	C (Traditional C)
8 bits	CHARACTER	char
8 bits	TINY_INTEGER	signed char
16 bits	SHORT_INTEGER	short
32 bits	POSITIVE	int
32 bits	NATURAL	int
32 bits	INTEGER	int
32 bits	SHORT_FLOAT (floating point)	float
64 bits	FLOAT (floating point)	double
32 bits	FIXED_POINT_TYPE	float
32 bits	ENUMERATION TYPES	int
32 bits	BOOLEAN	int
Variable†	ARRAY TYPES (Interface by the address of the first element)	array types
Variable†	STRING TYPES (Interface by the address of the first character)	char*
Variable†	RECORD TYPES (Interface by access types)	struct
32 bits	ACCESS TYPES	pointer

† sizes of the the elements must be the same in both Ada and C

Table 3.4. The data type and alignment size using in Ada and C/C++

3.4 *Implementation Ada/ObjectStore Facilities*

Verdix Ada shows how to create parallel data types and declare external subprograms. The prototype of Ada/ObjectStore, a high level view, then shows how to design a simple and efficient model: basic persistent Ada instances (manual schema generation). This work continues the approach of manual schema generation from the prototype of Ada/ObjectStore and expands its capabilities. Another concept of Ada/ObjectStore designed is that all `PRAGMA INTERFACE` statements are put in the package body. As we know the Ada package, one of the fundamental program units, permits a user to encapsulate a group of logically related entities. As such, they directly support the software principles of data abstraction and information hiding. The specification of packages forms the programmer's contract with the package client. A client never needs to see the package body and does not need to know how the functions work. Some functions in ObjectStore need to be handled before being called from Ada because of incompatible parameters, such as STRING type. In these cases, an intermediate subprogram is needed. However, most functions in ObjectStore can be directly called from Ada using `PRAGMA INTERFACE` without special handling. Ada/ObjectStore follows Ada's software design principles that enforces a clean interface of a package specification to clients; all `PRAGMA INTERFACE` statements are collected in the package body even without an intermediate subprogram.

After the functions of ObjectStore were analyzed, there are some new operations and types that need to be addressed. Those are described as follows:

- New types from the ObjectStore:

  The new types from ObjectStore, such as `os_collection*` or `os_cursor*` which are pointer types, are implemented the same as Ada/ObjectStore: all types of pointers in ObjectStore are handled in Ada as a new integer. Persistent objects are manipulated by ObjectStore. Ada acts like a temporary holder that receives the pointer from and sends it back to ObjectStore. Therefore, these types are declared as a user defined type and its base type is integer. For example `OS_COLLECTION` is defined as follows:

```
type OSTORE_OPAQUE is new INTEGER;
type OS_COLLECTION is new OSTORE_OPAQUE;
```

- Procedure variables:

  Ada does not support procedure variables (23). It is difficult for Ada is to simulate a parallel function in which C can combine procedure variables and a bit-wise operator in a simple statement. However, for this simple statement, Ada needs a lot of works to simulate it. For

example, in the function OS_COLLECTION_CHANGE_BEHAVIOR, one of its parameters is BEHAVIOR and its type is OS_UNSIGNED_INT32. In C, it is defined a type as follow:

```
typedef enum os_collection_behavior {
        os_collection_maintain_cursors=1,
        os_collection_allow_duplicates=1<<1,
        os_collection_signal_duplicates=1<<2,
        os_collection_allow_nulls=1<<3,
        os_collection_maintain_order=1<<4,
      } os_collection_behavior;
```

When the BEHAVIOR is sent to ObjectStore, it is simple to use randomly combined variables in the parameter list to compute a OS_UNSIGNED_INT32's value, such as:

```
os_collection_change_behavior(os_coll,
           os_collection_maintain_cursors   |
           os_collection_allow_duplicates   |
           os_collection_maintain_order )
```

Fortunately, in this special case each procedure produces a value that is power of 2, and the bit-or operation then is fundamentally the same as the add operation. Two ways can be used to simulate this behavior in Ada:

1. Declaring global constants:

```
MAINTAIN_CURSORS   : constant OS_UNSIGNED_INT32 := 1;
ALLOW_DUPLICATES   : constant OS_UNSIGNED_INT32 := 2;
SIGNAL_DUPLICATES  : constant OS_UNSIGNED_INT32 := 4;
ALLOW_NULLS        : constant OS_UNSIGNED_INT32 := 8;
MAINTAIN_ORDER     : constant OS_UNSIGNED_INT32 := 16;
```

   The advantage of this way is that when the value of BEHAVIOR is sent to ObjectStore it is easy to perform a bit-or operation by adding constant integer variables. The above function could be simulated as follows:

```
os_collection_change_behavior(OS_COLL,
          MAINTAIN_CURSORS   +
          ALLOW_DUPLICATES   +
          MAINTAIN_ORDER )
```

   However, the disadvantage of this way is the value of BEHAVIOR in Ada received from ObjectStore. The value shown to the user is just an integer such as, for example, 1. This is an unfriendly user interface. The interface should provide a friendly user interface so that the value shown to user should be an enumeration value such as, for example, MAINTAIN_CURSORS. Ada does not allow a type overloading in the same scope. In this

case, if MAINTAIN_CURSORS is declared as constant OS_UNSIGNED_INT32, Ada does not allow it to be declared as a value in an enumeration type in the same scope.

2. Declaring an enumeration type:

```
type OS_COLLECTION_BEHAVIOR is (MAINTAIN_CURSORS,
                                ALLOW_DUPLICATES,
                                SIGNAL_DUPLICATES,
                                ALLOW_NULLS,
                                MAINTAIN_ORDER);
```

To implement in this way, an additional subprogram is needed to parse an input string and then to compute the result of BEHAVIOR before it is sent to ObjectStore. The same function simulated to C is as follows:

```
os_collection_change_behavior(OS_COLL,
        "MAINTAIN_CURSORS
         ALLOW_DUPLICATES
         MAINTAIN_ORDER" )
```

The disadvantage of this method is that it involves another subprogram and surely takes more time. However, the advantage is that when the value of BEHAVIOR in Ada is received from ObjectStore, it can be easily transferred to a meaningful word such as, for example, an enumeration value–MAINTAIN_CURSORS.

User-defined enumeration types help to make programs more readable, understandable, and maintainable. The maintainability of software written in Ada is one positive aspect in comparison to C. Considering the benefits of user-defined enumeration types, declaring an enumeration type is a better approach than the declaring global constants.

- Naming convention:

  In order to implement the same functionality that ObjectStore has from Ada, the subprogram's name in Ada should be exactly the same as in the ObjectStore. For example, the function os_collection_create in ObjectStore is implemented in the function OS_COLLECTION_CREATE in Ada. The only difference is that the former is in lowercase, but the latter is in uppercase.

After these consideration mentioned above have been done, the functions in the Ada/ObjectStore corresponding to the ObjectStore are implemented readily, mapping functions in ObjectStore with Ada subprograms.

3.5 Testing of Ada/ObjectStore

The purpose of the testing is to check the execution of the software against the requirements in the specifications of Objectstore Reference Manual. Testing is done by a group of functions that have been implemented.

3.5.1 Testing Ada/ObjectStore functionality. Testing of Ada/ObjectStore is based on subprograms implemented. However, test programs are designed from a simple program which works correctly and then inserts a new subprogram when possible in a suitable position. A software testing method is used: white box testing that requires execution each statement at least once. All results are compared in compliance with the ObjectStore Reference Manual.

3.5.2 Performance Testing. Cattel points out that "The most accurate measure of performance for engineering applications would be to run an actual application, representing the data in the manner best suited to each potential DBMS." (9:364) He summarizes the three most important measures of performance in an object-oriented DBMS as:

- Lookup and Retrieval. Look up and retrieve an object given its identifier.
- Traversal. Find all objects in the hierarchy of a selected object.
- Insert. Insert objects and their relationships to other objects.

Berre and Anderson's HyperModel benchmark (5) presents a similar approach to performance measurement. In addition to the operations proposed by Cattel, the HyperModel benchmark includes:

- Sequential Scan. Visit each object in the database sequentially.
- Closure Operations. Perform operations on all objects reachable by a certain relationship from a specified object.
- Open-and-Close. Time to open and close the database.

We want to compare the performance of Ada/ObjectStore to ObjectStore. The Sun operating system provides profiling options which are implemented by the compiler. This profiling provides detailed timing and usage statistics from processes specified by the user. A program (CommandStats.c) (11:28) is written for gathering statistics. CommandStats makes calls to the Unix functions **getrusage** and **gettimeofday** and calculates processing time. Time is measured in CPU, user, and elapsed time. The CPU time is the total amount of time spent executing in system mode. The user time is the total amount of time spent executing in user mode. The elapsed time is the total

amount of time spent executing a process. Two CommandStats are needed in a testing program, and they are put before and after the module which is measured. Because CommandStats.c is a C program, it can directly be used in testing program written in C. However, a testing program written in Ada can not directly call CommandStats.c. An interface package, statis_ada.a and a C program, stat.c, are required to perform the job of accessing CommandStats.c

Performance testing compares the differences in access time between programs written in Ada/ObjectStore and ObjectStore that access the same ObjectStore database. The two programs are designed to correspond to each other as closely as possible. For example, the test sequence in Ada/ObjectStore is to call `OS_CURSOR_CREATE` and at same place in the test sequence for ObjectStore a corresponding function `os_cursor_create` is used.

The performance comparisons are conducted using these guidelines as mentioned. One exception is that the `Closure Operations` in the HyperModel benchmark are not measured because the ObjectStore C library interface does not provide the capability of relationships. Also, because the performance compares the difference between Ada/ObjectStore and ObjectStore C library interface, a complicated data model is not involved currently in this performance testing. Two kinds of simple objects were created to collect statistical data. They are classified in two test groups.

1. A single object:

   A single integer "count" is stored in ObjectStore. Three test programs were implemented; two of them are written in Ada accessing C library and C++ library (mangled name), and the third is written in C. The test programs, hello_ost.a(c) in different directories provide the performance testing for accessing this integer "count". (Appendix B.14 and B.12)

2. A compound object:

   Two record data structures `LINK_NOTE` and `NOTE_COL`, which contain 108 and 104 bytes respectively, were implemented. The database `bnote.db` was created to store 10,000 `LINK_NOTE`s for Ada/ObjectStore test programs and the database `cnote.db` was create to store 10,000 `NOTE_COL`s for Ada/ObjectStore collection test programs. The data structure for `LINK_NOTE` is declared as follows:

```
struct  link_note
 {
    int          priority;  /*  4 bytes */
    char         name[20];  /* 20 bytes */
    char         note[80];  /* 80 bytes */
    struct link_note *next; /*  4 bytes */
  };
```

The data structure for NOTE_COL is almost the same as LINK_NOTE except the field *next. The member of struct *next is no longer needed because of the ObjectStore collection that already provides facilities to manipulate its elements. The same structures were defined in Ada as follows:

```
type LINK_NOTE;
type LINK_NOTE_PTR is access LINK_NOTE;
type LINK_NOTE is
record
        priority: integer;
        name    : string(1..20);
        note    : string(1..80);
        next    : link_note_ptr;
end record;
```

Two test programs, adaobj.a and adaobj.c, also perform the job of testing basic functions for Ada and C respectively (Appendix B.2 and B.6). The extended functions "collection" is tested by adacol.a and adacol.c (Appendix B.4 and B.8). The performance is measured in the area of initializing, lookup and retrieval, sequential scan, and opening and closing a database.

- Initializing a database. The function initial_db inserts 10,000 objects in the database.
- Opening and closing a database. The function DATABASE_OPEN_CLOSE measures the time required to open and close a database 10 times.
- Look up and retrieve an object from the database. The function DATA_RETRIEVE searches the database until it finds the specified object and then displays it.
- Sequential scan. The DATA_SCAN procedure finds the database root and then gets objects one by one in the database and displays them.

In order to observe the performance change after both Ada and C added the facility of data persistence, program purobj.a and purobj.c were implemented for Ada and C, respectively (Appendix B.9 and B.10). The two programs are the same as adaobj.a and adaobj.c except that all functions calling the database are removed. That means all objects created are transient only.

3.6 Summary

The Ada language provides abilities to interface different languages. The Ada compiler gives a specific and detailed method to implement variables and subprograms in the interface. Ada/ObjectStore is a milestone for Ada accessing a database. We can expand its functions and implement those functions in a set of abstract modules, which is Ada's package. The abstract

module is used as an interface layer between the application module, which is Ada programs, and the concrete module, which is pure ObjectStore functions. Ada programmers create Ada programs as they used to be. They don't need to know how the persistent data are handled by a concrete module. The collection facilities in Ada/ObjectStore enhance a programmer's designing abilities and, at the same time, the maintainability is accomplished in short and simple statements. The testing of Ada/ObjectStore is done by functionality and performance. After the testing, we can compare the difference between Ada/ObjectStore and ObjectStore using the C library interface.

IV. Results Analysis

4.1 Overview

The primary objective of this thesis is to show that an interface of Ada and ObjectStore can provide almost the same functionality and performance as ObjectStore. The database was designed in the same way as in ObjectStore and it is described in Section 3.5.2. Because all objects are stored into ObjectStore, not Ada, so as long as an object has crossed the boundary of interface into ObjectStore, it is handled by ObjectStore. Performance is the vital factor in judging the difference between Ada/ObjectStore and ObjectStore. However, because the performance does not measure the efficiency of ObjectStore itself in managing objects, two simple objects are required to be created and stored in these tested databases. This chapter points out the different performances for Ada and C in manipulating the database.

4.2 Performance Comparison of Ada/ObjectStore and ObjectStore

The results of the performance testing where a single integer is stored in ObjectStore are shown in Tables 4.1 and 4.2. From these results, some conclusions can be drawn. Table 4.1 and Table 4.2 show that when manipulating a single integer, if the integer was retrieved and modified once, Ada/ObjectStore's performance is much slower than C/ObjectStore's. However, if it runs 100 times, Ada/ObjectStore's performance is close to C/ObjectStore's. Table 4.3 shows the difference between two Ada/ObjectStore programs, that one run using the C++ mangled interface is better than one run using the C library interface. However, the difference in performance between these two programs decreases when they run 100 times.

The results of the performance for testing a database created by ObjectStore using a linked list and ObjectStore using collections are shown in Table 4.4 and 4.5. From the testing results of initialization and opening and closing a database, Ada seems doing a better job on dynamic storage allo-

Criteria Tested	Resource Measured	Application Programs		Percent Change
		hello_ost.a	hello_ost.c	
Run Once	User time (seconds)	0.032	0.038	-15.8
	CPU time (seconds)	0.345	0.108	+219.4
	Elapsed time (seconds)	0.778	0.332	+134.3
	Page Faults without I/O	215.883	142.000	+52.0
Run 100 Times	User time (seconds)	0.680	0.635	+7.1
	CPU time (seconds)	1.853	1.522	+21.7
	Elapsed time (seconds)	13.611	12.666	+7.5
	Page Faults without I/O	1007.333	939.167	+7.3

Table 4.1. Benchmark performance results for hello_ost.a and hello_ost.c

Criteria Tested	Resource Measured	Application Programs		Percent Change
		hello_ost.a	hello_ost.c	
Run Once	User time (seconds)	0.043	0.038	+13.2
	CPU time (seconds)	0.195	0.108	+80.6
	Elapsed time (seconds)	0.579	0.332	+74.4
	Page Faults without I/O	211.000	142.000	+48.6
Run 100 Times	User time (seconds)	0.677	0.635	+6.6
	CPU time (seconds)	1.713	1.522	+12.5
	Elapsed time (seconds)	13.275	12.666	+4.8
	Page Faults without I/O	1003.000	939.167	+6.8

Table 4.2. Benchmark performance results for hello_ost.a and hello_ost.c (C++ mangling interface)

Criteria Tested	Resource Measured	Application Programs		Percent Change
		hello_ost.a C++ Library	hello_ost.a C Library	
Run Once	User time (seconds)	0.043	0.032	+34.4
	CPU time (seconds)	0.195	0.345	-43.5
	Elapsed time (seconds)	0.579	0.778	-25.6
	Page Faults without I/O	211.000	215.883	-2.3
Run 100 Times	User time (seconds)	0.677	0.680	-0.4
	CPU time (seconds)	1.713	1.353	-7.6
	Elapsed time (seconds)	13.275	13.611	-2.5
	Page Faults without I/O	1003.000	1007.333	-0.4

Table 4.3. Benchmark performance results for hello_ost.a accessing C++ and C library interface

Criteria Tested	Resource Measured	Application Programs		Percent Change
		adaobj.a	adaobj.c	
Initialize (10,000 Objects inserted)	User time (seconds)	1.233	0.966	+27.6
	CPU time (seconds)	0.321	0.330	-2.7
	Elapsed time (seconds)	6.502	4.760	+36.6
	Page Faults without I/O	290.500	319.000	-8.9
Open & Close	User time (seconds)	0.076	0.059	+28.8
	CPU time (seconds)	0.070	0.119	-41.2
	Elapsed time (seconds)	0.388	0.269	+44.2
	Page Faults without I/O	60.000	60.000	0.0
Look up/Retrieve	User time (seconds)	0.023	0.013	+76.9
	CPU time (seconds)	0.068	0.083	-18.1
	Elapsed time (seconds)	0.095	0.097	-2.1
	Page Faults without I/O	134.500	134.000	+0.4
Sequential Scan (With Output to Screen)	User time (seconds)	4.200	0.641	+555.2
	CPU time (seconds)	30.295	14.576	+107.8
	Elapsed time (seconds)	181.928	144.643	+25.8
	Page Faults without I/O	265.667	267.875	-0.8
Sequential Scan (Without Output To Screen)	User time (seconds)	0.046	0.029	+58.6
	CPU time (seconds)	0.146	0.175	-16.6
	Elapsed time (seconds)	0.195	0.203	-3.9
	Page Faults without I/O	265.000	265.375	-0.1

Table 4.4. Benchmark performance results for adaobj.a and adaobj.c

Criteria Tested	Resource Measured	Application Programs		Percent Change
		adacol.a	adacol.c	
Initialize (10,000 Objects inserted)	User time (seconds)	2.771	2.633	+5.2
	CPU time (seconds)	0.254	0.419	-39.4
	Elapsed time (seconds)	6.671	7.464	-10.6
	Page Faults without I/O	347.000	360.000	-3.6
Open & Close	User time (seconds)	0.076	0.070	+8.6
	CPU time (seconds)	0.06	0.145	-58.6
	Elapsed time (seconds)	0.289	0.285	+1.4
	Page Faults without I/O	60.000	60.000	0.0
Look up/Retrieve	User time (seconds)	0.277	0.266	+4.1
	CPU time (seconds)	0.202	0.186	+8.6
	Elapsed time (seconds)	0.476	0.451	+5.5
	Page Faults without I/O	286	286	0.0
Sequential Scan (With Output to Screen)	User time (seconds)	4.520	1.035	+336.7
	CPU time (seconds)	30.625	13.305	+130.2
	Elapsed time (seconds)	191.779	128.191	+49.6
	Page Faults without I/O	299.833	299.500	+0.1
Sequential Scan (Without Output To Screen)	User time (seconds)	0.113	0.106	+6.6
	CPU time (seconds)	0.033	0.026	+26.9
	Elapsed time (seconds)	0.149	0.133	+12.0
	Page Faults without I/O	37.250	34.000	+9.6

Table 4.5. Benchmark performance results for adacol.a and adacol.c

Criteria Tested	Resource Measured	Application Programs		Percent Change
		purobj.a	purobj.c	
Initialize (10,000 Objects inserted)	User time (seconds)	0.267	0.195	+36.9
	CPU time (seconds)	0.147	0.218	-32.6
	Elapsed time (seconds)	0.414	0.412	+0.5
	Page Faults without I/O	0.000	0.000	0.0
Look up/Retrieve	User time (seconds)	0.007	0.005	+40.0
	CPU time (seconds)	0.005	0.005	0.0
	Elapsed time (seconds)	0.012	0.011	+9.1
	Page Faults without I/O	0.000	0.000	0.0
Sequential Scan (With Output to Screen)	User time (seconds)	4.267	1.397	+205.4
	CPU time (seconds)	34.394	9.206	+273.6
	Elapsed time (seconds)	335.349	141.660	+136.7
	Page Faults without I/O	295.500	306.000	-3.4
Sequential Scan (Without Output To Screen)	User time (seconds)	0.015	0.015	0.0
	CPU time (seconds)	0.033	0.038	-13.2
	Elapsed time (seconds)	0.050	0.053	-5.7
	Page Faults without I/O	294.000	295.000	-0.3

Table 4.6. Benchmark performance results for purobj.a and purobj.c

Criteria Tested	Resource Measured	Application Programs		Percent Change
		adaobj.a	purobj.a	
Initialize (10,000 Objects inserted)	User time (seconds)	1.235	0.267	+361.8
	CPU time (seconds)	0.321	0.147	+118.4
	Elapsed time (seconds)	6.502	0.414	+1470.5
	Page Faults without I/O	290.500	0.000	0.0
Look up/Retrieve	User time (seconds)	0.023	0.007	+228.6
	CPU time (seconds)	0.068	0.005	+1260.0
	Elapsed time (seconds)	0.095	0.012	+691.7
	Page Faults without I/O	134.500	0.000	0.0
Sequential Scan (With Output to Screen)	User time (seconds)	4.200	4.267	-1.6
	CPU time (seconds)	30.295	34.394	-11.9
	Elapsed time (seconds)	181.928	335.349	-45.7
	Page Faults without I/O	265.667	295.500	-10.1
Sequential Scan (Without Output To Screen)	User time (seconds)	0.046	0.015	+206.7
	CPU time (seconds)	0.146	0.033	+342.4
	Elapsed time (seconds)	0.195	0.050	+290.0
	Page Faults without I/O	265.000	294.000	-9.9

Table 4.7. Benchmark performance results for adaobj.a and purobj.a

Criteria Tested	Resource Measured	Application Programs		Percent Change
		adaobj.c	purobj.c	
Initialize (10,000 Objects inserted)	User time (seconds)	0.966	0.195	+395.4
	CPU time (seconds)	0.330	0.218	+51.4
	Elapsed time (seconds)	4.760	0.412	+1055.3
	Page Faults without I/O	319.000	319.000	0.0
Look up/Retrieve	User time (seconds)	0.013	0.005	+160.0
	CPU time (seconds)	0.083	0.005	+1560.0
	Elapsed time (seconds)	0.097	0.011	+781.7
	Page Faults without I/O	134.000	0.000	0.0
Sequential Scan (With Output to Screen)	User time (seconds)	0.641	1.397	-54.1
	CPU time (seconds)	14.576	9.206	+58.3
	Elapsed time (seconds)	144.643	141.660	+2.1
	Page Faults without I/O	267.875	306.000	-12.5
Sequential Scan (Without Output To Screen)	User time (seconds)	0.029	0.015	-80.7
	CPU time (seconds)	0.175	0.038	+360.5
	Elapsed time (seconds)	0.203	0.053	-61.7
	Page Faults without I/O	265.375	295.000	-10.0

Table 4.8. Benchmark performance results for adaobj.c and purobj.c

cation and handling a pointer; the CPU time of Ada/ObjectStore is faster than C/ObjectStore. The sequential scan consistently shows that moving a pointer along the linked list of Ada/ObjectStore is indeed faster than what C/ObjectStore does. The performance that varies most significantly in both Tables is in sequential scan. CPU time and User time of Ada/ObjectStore are over 2 times slower than that of C/ObjectStore. In the testing subprogram of sequential scan, timing is measured from traversing the link list and printing every note when it is traversed. In order to find out why the performance varies so significantly, a small change in testing subprogram of sequential scan was made. The testing only let processes traverse notes, but every note traversed does not have to be printed out to the screen. The results of the performance tests are shown in Table 4.4 and Table 4.5. It shows the performance is not much different between Ada/ObjectStore and C/ObjectStore. The main factors affecting the result of performance still depends on the two languages' own abilities. The performance of TEXT_IO in Ada is slower than the performance of `printf` in C.

Tables 4.6 - 4.8 show the different performance of Ada and C in manipulating transient and persistent data. Table 4.6 shows the test of a transient object only. Ada still does well in storage allocation. The pointer moving along the linked list is good, too. Consistently this shows that Ada is indeed good at handling the pointer moving. Both Ada and C performances decrease in order to provide data persistence. However, Table 4.7 and Table 4.8 show that the performance degradation is not much different between Ada and C when accessing ObjectStore.

Files		Size in Kbytes		Percent
		Written in Ada	Written in C	Change (%)
Hello_ost	Source file	1.5	1.0	+50.0
	Excutable file	1,515.5	1,327.1	+14.2
Adaobj	Source file	8.5	8.5	0
	(Excutable file)	1,548.3	1,343.5	+15.2
Adacol	Source file	9.6	8.8	+9.1
	(Excutable file)	2,400.2	2,195.5	+9.3

Table 4.9. Comparison of file size written in Ada and C (static binding)

Space usage is worthy of some examination. Source programs, executable programs and databases are observed. As we understand, source files are heavily dependent on the programming behavior of programmers. The size of executable programs are dependent on compilers used. Due to compiler limitations, we needed to use static binding when performing the link. Table 4.9 shows that both application programs' sizes and executable programs' sizes in Ada are larger than C's. Strong type checking and the interface programs added overhead can probably explain this.

4.3 Problems Encountered

One of the objectives was to expand ObjectStore functions in Ada/ObjectStore as much as possible. Some difficulties arose because of the quite different syntax of the two languages. Most problems are categorized by understanding ObjectStore C library functions, limitations of the interface between Ada and C, and losing the ability of the programming tool–the debugger.

4.3.1 Debugger. Most programs have errors in syntax and semantics. Ada is a strongly typed language. It can find syntactic problems at compile time. However, at run time, semantic errors need a debugger to trace out. The debugger can detect and report on a wide variety of problems, including variables that are used before they are set and after, and arguments in functions changed after functions are called. Verdix has a debugger, a.db, but it seems that it can not debug programs written in Ada/ObjectStore. When the debugger was executed, it would stop at statement "DATABASE_OPEN" and give an error message as follows:

```
''Segmentation fault'' I/O error: trying to read u.u_code [Unix errno: 5]
--> Segmentation Violation (SIGSEGV) code: 255 (u_code: -1)
```

All the Ada/ObjectStore programs I wrote are small test programs. Usually they are not over 300 lines. However, in a practical application system, programs are commonly over several

thousand lines. Without a debugger, a big obstacle lays in the way of developing a system using Ada/ObjectStore.

4.3.2 Understanding ObjectStore. ObjectStore's documentation mainly describes C++ library functions, and some of them have examples shown in the User Guide. C library functions are listed in the Reference Manual and the reader is referred to see the C++ analogous functions for detailed information. Theoretically, C++ and C are analogous. However, there are some cases where C is not quite compatible with C++ (11:41). Casting is used heavily in C, but not in C++ because C++ has a stronger type checking ability. OS_CURSOR is another example. In the C++ library, OS_CURSOR is declared a class and a constructor OS_CURSOR. To create a OS_CURSOR is to initialize the class OS_CURSOR with the required argument, OS_COLLECTION. An example is "os_cursor cur(os_collection)". There are no pointer values returned. However, OS_CURSOR_CREATE in the C library has quite a different syntax in which it will simply return a pointer that points to one element associated with its collection.

Database_create and **database_lookup** do not work with static binding. To remedy this, a program was implemented to perform only one job, creating a database. After the program created the database, the performance testing was then continued.

Error messages indicated what errors occurred, but the specific information related to the error is not fully explained. As previously stated, **database_create** does not work in static binding; the error message indicates "some kinds of initialization needed to be done", but it does not show what kinds of initialization and how to initialize.

4.3.3 Interface Limitations. Besides the limitation of Ada/C interface for handling procedure variables, the Query facility in ObjectStore has a pre-analyzed query. To use this facility, it requires three steps (17:150):

1. analysis of the query expression,
2. binding of the free variable and function references in the query, and
3. actual interpretation of the bound query

The problem happens at the **os_keyword_arg_list*** when processes go to the actual interpretation of the bound query. The function os_bound_query takes two arguments: a pre-analyzed query, and a keyword_arg list. It is defined as follows:

```
extern os_bound_query* os_bound_query_create(
   os_coll_query*,/* the query to bind */
```

```
    os_keyword_arg_list*
     /* the argument list with binding for free vars */
);
```

The os_keyword_arg_list is expressed in the following form:

```
(
  keyword_arg-expression,
    ...,
    ...,
  keyword_arg-expression
)
```

Because the arguments, consisting by keyword_args of keyword_arg_list is not fixed, to simulate the same functionality in Ada may be complicated and inefficient.

4.4 Summary

Performance tests were performed for two kinds of areas using the Ada/ObjectStore and the functions of ObjectStore C library. A simple data structure was implemented to measure the performance. The result shows that there is not much difference when comparing the two languages that interface with ObjectStore; the difference still depends on the languages own properties. Some problems were encountered in the interfacing limitation that exists between the two language. Some problems were related to inadequacies in ObjectStore documentation. These difficulties affect the effort of implementing Ada/ObjectStore functions.

V. Conclusions and Recommendations

5.1 Overview

This chapter summarizes the activities in Ada accessing the ObjectStore database management system. Most basic and collection functions of the ObjectStore were implemented in several packages written in Ada, called Ada/ObjectStore. The functionality of Ada/ObjectStore are analyzed. Some advantages and disadvantage are discussed in the conclusions. Finally, recommendations are presented for future research to complete the interface.

5.2 Summary of Research

In the activities of this thesis, the ObjectStore functions were first familiarized. Secondly, parallel data types in Verdix Ada and in ObjectStore were examined. Finally, the initial prototype Ada/ObjectStore created by Object Design, Inc., was extended by using functions in the ObjectStore C library.

The packages of Ada/ObjectStore are implemented in a corresponding way to the functions in the ObjectStore C library. Most functions in the ObjectStore C library can be directly accessed from Ada without any change, but some of them require an intermediate level to handle incompatible types. However, the limitation of the interface implemented from Ada to C still strictly depends on the both languages' properties. For example, C provides procedure variables and shift operators, but Ada does not. C provides simple bitwise binary logical operators, but standard Ada does neither. Verdix Ada provides a bitwise function in a package, but it is implementation-defined and its binary operations are limited by Ada's syntax can not be expressed as simply as C does.

To compare the performance of Ada/ObjectStore with C/ObjectStore, several test programs were written in Ada and in C. Timing routines were instrumented in the code to measure the time required to access the ObjectStore commands called from Ada and C. The test programs designed were based on testing database functionality.

5.3 Conclusions

The objectives of this thesis were to implement an interface that has functional completeness. Moreover, the performance of Ada/ObjectStore should not be much different than C/ObjectStore. Some problems arose and these problems may affect functional completeness. The following discussion points out the advantages and disadvantages of this interface.

5.3.1 Data Persistence. The objective of data persistence is achieved in the interface. As described in Chapter 2, Ada does not support data persistence. However, ObjectStore, a DBMS using a C/C++ library interface, provides the ability to handle persistent data. Ada/ObjectStore accesses the database, which is managed by the ObjectStore, in exactly the same way as the C/ObjectStore does; the subprogram in the programming language can handle persistent and transient data without difference. All persistent data are managed by ObjectStore, which provides data management abilities.

Data persistence gives Ada programmers great benefits. First, they do not need to write program I/O statements. Second, they do not need to write a lot of statements for mapping values between transient and persistent data. Third, because program sizes are decreased, software productivity and maintainability are increased.

5.3.2 Reliability, Maintenance, and Efficiency. The Ada reference manual (1) points out "Ada was designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency".

Ada is a strongly typed language. This is based on the design goal "program reliability and maintenance"(1). In order to achieve the reliability of the interface, certain rules must apply to ensure type safety that are described in section 3.3.2. For example, all variables need to be explicitly declared and their type specified. The compilers can then check to see that operations on these variables are compatible with the properties of their type. Because variables are safely manipulated in Ada, program reliability is maintained.

Ada is proud of the standard coding format and this is acknowledged by all who have ever seen the Ada program. In contrast, C/C++ features ease of writing rather than ease of reading. The data persistence, which allows programming in clarity and simplicity, increases the advantages of the maintenance of the Ada programs.

To achieve efficiency, Ada was constructed and carefully examined in the light of present implementation techniques. Any proposed construct whose implementation was unclear or that required excessive machine resources was rejected (1). This can be demonstrated from Table 4.4 in which the CPU time of Ada is faster than that of C. However, in order to add the ability of data persistence, which is provided by a database, some trade offs in efficiency must be faced. But the efficiency in Ada is not much worse than in C when they access ObjectStore.

5.3.3 Data Abstraction. Ada is not truly an OOPL, but the Ada contains most OOPL concepts, namely the encapsulation (package), information hiding (private types and package bodies),

and concurrent processing (task). In the interface, any type declared in Ada can be implemented with a corresponding base type in C. The Ada programmer has almost no limit to model the real world objects using Ada's data abstraction. Information hiding is achieved via private types. Because data type is needed to be persistent in ObjectStore, the data type defined in Ada must be converted to an **os_typespec** before it can accompany its object stored to and retrieved from ObjectStore. The **pragma INTERFACE** statement for accessing **os_typespec** is allowed at the place after the data type are fully defined, and, if a private data type is defined, the **pragma INTERFACE** must appear in the same package specification after the type is fully declared.

5.4 Recommendations for Future Research

This thesis extended the prototype of the Ada/ObjectStore, extending its functional completeness. But the goal was not completely achieved. Most functions of **collection** in the ObjectStore C library can be accessed by the Ada now, but still a lot of works need to be done. These are the transparent interface to Ada programmers, exception handling, and version management.

5.4.1 Transparency. Ada/ObjectStore is not completely transparent to Ada programmers. In order to generate a parallel data structure in the ObjectStore a manual schema must be generated; a C macro facility is currently provided. Moreover, an **INTERFACE** call must be put in the main procedure before calling any function that is associated with os_typespec. The query facility of Ada/ObjectStore is another example. The query string, for example "strcmp(name,"Mike") == 0)" for querying a string type or "age == 35" for querying a scalar type, is still the C language's syntax, "strcmp" and "==" in this example. To achieve the transparency, some intermediate subprogram needs to be created and act as preprocessor or translator.

5.4.2 Exception Handling. Both Ada and the ObjectStore C library provide abilities for handling exceptions. The exception facility is very important for dealing with errors or other exceptional situations during program execution. An exception can be raised by a raise statement or operations that propagate the exception. When an exception arises, control can be transferred to a user-provided exception handler. The interface of declaring exceptions is not completely implemented.

5.4.3 Version Management. Version management is very important in the area of computer-aided design applications today. These applications need to increasingly support cooperative work by a number of engineers on the same design. Ada/ObjectStore does not yet

provide version management facilities. That means Ada/ObjectStore is not currently good in CAD applications or other applications that need data checked out for extended periods of time.

5.4.4 Variant Records. Ada contains most of the concepts of the object oriented philosophy as described in 5.3.3. These features make Ada very close to an OOPL. However, the only major object oriented concepts not supported by Ada are dynamic binding and inheritance (15). Rumbaugh (19) and Leopold (15) pointed out that using variant records, Ada can have the ability of single inheritance. A variant record is a record structure and contains a discriminant that distinguishes the alternate forms of the record. C does not provide variant records in the same way. However, the union type in C can perform the same capability in Ada. Therefore, the ability of variant records should be maintained in this interface.

5.5 Summary

Although Ada/ObjectStore is not completely implemented, the results that what have been developed are satisfactory. The performance of Ada/ObjectStore does not differ much with respect to C/ObjectStore, but the enhancement of abilities to manipulate persistent data is a great advantage for Ada. Many problems remain. Some of them are the completeness of functionality, and some are limitations of the languages. Ada has been acknowledged as a good language in maintainability. Binding to the OODBMS gives Ada great potential in the development and maintenance of complex, data intensive, engineering applications.

Appendix A. Raw Performance Test Results

Test Program: hello_ost.a (C library interface)

Command	User Time	CPU Time	Elapsed Time	Page Faults		Disk Blocks	
				with I/O	w/o I/O	In	Out
Run Once	0.030	0.360	0.786	0	213	0	0
	0.040	0.370	0.768	0	214	0	0
	0.040	0.290	0.728	0	214	0	0
	0.030	0.380	0.816	0	213	0	0
	0.040	0.320	0.800	0	227	0	0
	0.010	0.350	0.767	0	214	0	0
Run 100 Times	0.700	1.820	11.936	0	1005	0	0
	0.580	1.860	13.940	0	1005	0	0
	0.700	1.970	14.480	0	1005	0	0
	0.660	1.830	13.652	0	1017	0	0
	0.710	1.790	13.764	0	1006	0	0
	0.730	1.670	13.892	0	1006	0	0

Test Program: hello_ost.a (C++ library interface)

Command	User Time	CPU Time	Elapsed Time	Page Faults		Disk Blocks	
				with I/O	w/o I/O	In	Out
Run Once	0.040	0.210	0.631	0	221	0	0
	0.020	0.180	0.585	0	209	0	0
	0.060	0.250	0.776	0	209	0	0
	0.040	0.190	0.523	0	209	0	0
	0.050	0.190	0.475	0	209	0	0
	0.050	0.150	0.482	0	209	0	0
Run 100 Times	0.700	1.650	11.893	0	1001	0	0
	0.640	1.710	13.207	0	1001	0	0
	0.730	1.680	13.508	0	1001	0	0
	0.730	1.680	13.500	0	1013	0	0
	0.590	1.830	13.505	0	1001	0	0
	0.670	1.730	14.037	0	1001	0	0

Test Program: hello_ost.c

Command	User Time	CPU Time	Elapsed Time	Page Faults		Disk Blocks	
				with I/O	w/o I/O	In	Out
Run Once	0.040	0.130	0.439	0	142	0	0
	0.030	0.110	0.313	0	142	0	0
	0.030	0.140	0.289	0	142	0	0
	0.060	0.100	0.298	0	142	0	0
	0.040	0.070	0.331	0	142	0	0
	0.030	0.100	0.319	0	142	0	0
Run 100 Times	0.710	1.390	11.416	0	950	0	0
	0.580	1.620	13.762	0	934	0	0
	0.480	1.660	13.826	0	934	0	0
	0.600	1.670	14.624	0	881	0	0
	0.620	1.480	12.356	0	934	0	0
	0.660	1.530	11.884	0	947	0	0

Test Program: adaobj.a

Command	User Time	CPU Time	Elapsed Time	Page Faults with I/O	Page Faults w/o I/O	Disk Blocks In	Disk Blocks Out
Initialize	1.270	0.290	8.104	0	291	0	0
(10,000	1.230	0.350	9.144	0	284	0	0
Objects	1.270	0.350	5.733	0	291	0	0
inserted)	1.170	0.190	3.695	0	291	0	0
	1.200	0.260	6.560	0	291	0	0
	1.150	0.160	6.845	0	292	0	0
	1.440	0.490	6.086	0	293	0	0
	1.130	0.480	5.846	0	291	0	0
Open &	0.060	0.090	0.241	0	60	0	0
Close	0.100	0.060	0.246	0	60	0	0
	0.060	0.090	0.308	0	60	0	0
	0.070	0.080	0.425	0	60	0	0
	0.080	0.090	0.581	0	60	0	0
	0.090	0.050	0.770	0	60	0	0
	0.080	0.050	0.277	0	60	0	0
	0.070	0.050	0.258	0	60	0	0
Lookup/	0.020	0.080	0.092	0	134	0	0
Retrieve	0.020	0.070	0.095	0	134	0	0
	0.030	0.060	0.097	0	134	0	0
	0.020	0.070	0.100	0	134	0	0
	0.040	0.050	0.095	0	134	0	0
	0.010	0.080	0.094	0	134	0	0
Sequential	4.150	32.730	185.071	0	266	0	0
Scan (With	3.940	29.450	184.750	0	268	0	0
Output	4.270	29.570	185.649	0	265	0	0
to Screen)	4.220	29.490	190.724	0	266	0	0
	4.530	29.700	187.444	0	265	0	0
	4.090	30.830	157.928	0	265	0	0
Sequential	0.040	0.140	0.180	0	265	0	0
Scan (Without	0.070	0.200	0.264	0	265	0	0
Output	0.040	0.140	0.180	0	265	0	0
to Screen)	0.010	0.170	0.184	0	265	0	0
	0.060	0.120	0.181	0	265	0	0
	0.050	0.140	0.191	0	265	0	0
	0.040	0.140	0.185	0	265	0	0
	0.030	0.150	0.184	0	265	0	0

Test Program: adaobj.c

Command	User Time	CPU Time	Elapsed Time	Page Faults with I/O	Page Faults w/o I/O	Disk Blocks In	Disk Blocks Out
Initialize	0.960	0.270	4.880	0	319	0	0
(10,000	0.980	0.350	4.518	0	319	0	0
Objects	0.980	0.380	5.783	0	319	0	0
inserted)	0.970	0.230	4.344	0	319	0	0
	0.950	0.260	5.102	0	319	0	0
	0.980	0.220	4.331	0	319	0	0
	0.970	0.680	4.846	0	319	0	0
	0.940	0.250	4.280	0	319	0	0
Open &	0.040	0.130	0.263	0	60	0	0
Close	0.060	0.060	0.276	0	60	0	0
	0.040	0.160	0.271	0	60	0	0
	0.060	0.160	0.266	0	60	0	0
	0.070	0.130	0.282	0	60	0	0
	0.070	0.070	0.256	0	60	0	0
	0.070	0.120	0.274	0	60	0	0
	0.060	0.120	0.265	0	60	0	0
Lookup/	0.000	0.090	0.092	0	134	0	0
Retrieve	0.010	0.080	0.095	0	134	0	0
	0.010	0.090	0.097	0	134	0	0
	0.000	0.090	0.100	0	134	0	0
	0.020	0.070	0.095	0	134	0	0
	0.010	0.080	0.093	0	134	0	0
	0.030	0.070	0.105	0	134	0	0
	0.020	0.090	0.102	0	134	0	0
Sequential	0.620	14.520	162.831	0	265	0	0
Scan (With	0.550	13.850	151.141	0	269	0	0
Output	0.720	13.260	170.305	0	268	0	0
to Screen)	0.640	16.480	148.934	0	268	0	0
	0.550	14.280	149.505	0	268	0	0
	0.750	13.300	109.313	0	268	0	0
	0.750	17.070	113.976	0	268	0	0
	0.550	13.850	151.141	0	269	0	0
Sequential	0.030	0.140	0.174	0	267	0	0
Scan (Without	0.020	0.200	0.218	0	266	0	0
Output	0.050	0.160	0.214	0	265	0	0
to Screen)	0.040	0.160	0.203	0	265	0	0
	0.010	0.190	0.200	0	265	0	0
	0.020	0.200	0.215	0	265	0	0
	0.030	0.180	0.205	0	265	0	0
	0.030	0.170	0.195	0	265	0	0

Test Program: adacol.a

Command	User Time	CPU Time	Elapsed Time	Page Faults with I/O	Page Faults w/o I/O	Disk Blocks In	Disk Blocks Out
Initialize	2.750	0.240	6.161	0	347	0	0
(10,000	2.810	0.160	6.107	0	347	0	0
Objects	2.810	0.290	6.698	0	347	0	0
inserted)	2.720	0.350	6.484	0	347	0	0
	2.790	0.270	6.580	0	347	0	0
	2.880	0.270	6.564	0	347	0	0
	2.690	0.220	8.436	0	347	0	0
	2.720	0.230	6.401	0	347	0	0
Open &	0.070	0.080	0.368	0	60	0	0
Close	0.070	0.030	0.344	0	60	0	0
	0.070	0.070	0.307	0	60	0	0
	0.080	0.050	0.264	0	60	0	0
	0.060	0.080	0.263	0	60	0	0
	0.090	0.060	0.253	0	60	0	0
	0.070	0.060	0.252	0	60	0	0
	0.100	0.050	0.262	0	60	0	0
Lookup/	0.240	0.210	0.450	0	286	0	0
Retrieve	0.300	0.190	0.483	0	286	0	0
	0.320	0.150	0.463	0	286	0	0
	0.300	0.200	0.493	0	286	0	0
	0.230	0.260	0.490	0	286	0	0
	0.270	0.200	0.476	0	286	0	0
Sequential	4.550	29.380	193.678	0	300	0	0
Scan (With	4.870	30.730	196.403	0	300	0	0
Output	4.950	30.930	196.693	0	300	0	0
to Screen)	4.460	30.910	172.602	0	298	0	0
	4.650	30.130	195.041	0	300	0	0
	3.640	31.670	196.176	0	301	0	0
Sequential	0.120	0.030	0.148	0	47	0	0
Scan (Without	0.120	0.050	0.179	0	47	0	0
Output	0.110	0.040	0.150	0	34	0	0
to Screen)	0.120	0.020	0.141	0	34	0	0
	0.120	0.040	0.152	0	34	0	0
	0.100	0.050	0.145	0	34	0	0
	0.100	0.030	0.138	0	34	0	0
	0.110	0.030	0.140	0	34	0	0

Test Program: adacol.c

Command	User Time	CPU Time	Elapsed Time	Page Faults with I/O	Page Faults w/o I/O	Disk Blocks In	Disk Blocks Out
Initialize	2.770	0.740	7.295	0	360	0	0
(10,000	2.590	0.390	8.026	0	360	0	0
Objects	2.650	0.420	7.769	0	360	0	0
inserted)	2.560	0.400	6.554	0	360	0	0
	2.580	0.330	6.748	0	360	0	0
	2.650	0.400	8.062	0	360	0	0
	2.700	0.420	7.218	0	360	0	0
	2.560	0.250	8.036	0	360	0	0
Open &	0.080	0.160	0.313	0	60	0	0
Close	0.060	0.150	0.266	0	60	0	0
	0.080	0.110	0.263	0	60	0	0
	0.030	0.190	0.288	0	60	0	0
	0.070	0.160	0.280	0	60	0	0
	0.070	0.130	0.272	0	60	0	0
	0.090	0.100	0.303	0	60	0	0
	0.080	0.160	0.297	0	60	0	0
Lookup/	0.300	0.150	0.450	0	286	0	0
Retrieve	0.250	0.200	0.446	0	286	0	0
	0.210	0.240	0.453	0	286	0	0
	0.300	0.150	0.445	0	286	0	0
	0.280	0.170	0.450	0	286	0	0
	0.260	0.210	0.464	0	286	0	0
	0.280	0.170	0.449	0	286	0	0
	0.250	0.200	0.451	0	286	0	0
Sequential	0.910	11.650	125.425	0	299	0	0
Scan (With	1.110	11.670	125.996	0	299	0	0
Output	1.020	14.970	126.669	0	301	0	0
to Screen)	1.120	11.910	127.711	0	299	0	0
	0.900	13.020	130.031	0	300	0	0
	0.940	12.860	129.460	0	299	0	0
	1.160	14.910	130.299	0	299	0	0
	1.120	15.450	129.907	0	300	0	0
Sequential	0.110	0.010	0.127	0	46	0	0
Scan (Without	0.100	0.040	0.135	0	34	0	0
Output	0.130	0.010	0.140	0	34	0	0
to Screen)	0.110	0.020	0.128	0	34	0	0
	0.100	0.010	0.116	0	34	0	0
	0.100	0.030	0.125	0	34	0	0
	0.100	0.050	0.149	0	46	0	0
	0.100	0.040	0.145	0	34	0	0

Test Program: purobj.a

Command	User Time	CPU Time	Elapsed Time	Page Faults with I/O	Page Faults w/o I/O	Disk Blocks In	Disk Blocks Out
Initialize	0.260	0.150	0.408	0	0	0	0
(10,000	0.300	0.110	0.410	0	0	0	0
Objects	0.280	0.140	0.419	0	0	0	0
inserted)	0.270	0.140	0.417	0	0	0	0
	0.230	0.180	0.417	0	0	0	0
	0.260	0.160	0.424	0	0	0	0
Lookup &	0.010	0.010	0.013	0	0	0	0
Retrieve	0.010	0.000	0.015	0	0	0	0
	0.000	0.010	0.011	0	0	0	0
	0.010	0.000	0.011	0	0	0	0
	0.010	0.000	0.012	0	0	0	0
	0.000	0.010	0.011	0	0	0	0
Sequential	3.730	37.324	337.607	0	296	0	0
Scan (With	4.190	27.853	314.580	0	296	0	0
Output	4.330	27.003	426.015	0	296	0	0
to Screen)	3.460	48.485	355.470	0	295	0	0
	5.061	32.873	338.379	0	295	0	0
	4.830	32.823	240.044	0	295	0	0
Sequential	0.010	0.060	0.071	0	294	0	0
Scan (Without	0.020	0.020	0.042	0	294	0	0
Output	0.020	0.040	0.059	0	294	0	0
to Screen)	0.010	0.030	0.043	0	294	0	0
	0.020	0.020	0.042	0	294	0	0
	0.010	0.030	0.043	0	294	0	0

Test Program: purobj.c

Command	User Time	CPU Time	Elapsed Time	Page Faults with I/O	Page Faults w/o I/O	Disk Blocks In	Disk Blocks Out
Initialize	0.210	0.200	0.405	0	0	0	0
(10,000	0.200	0.210	0.408	0	0	0	0
Objects	0.210	0.200	0.412	0	0	0	0
inserted)	0.180	0.230	0.408	0	0	0	0
	0.190	0.230	0.418	0	0	0	0
	0.180	0.240	0.421	0	0	0	0
Lookup &	0.010	0.000	0.010	0	0	0	0
Retrieve	0.000	0.010	0.010	0	0	0	0
	0.010	0.000	0.011	0	0	0	0
	0.000	0.010	0.010	0	0	0	0
	0.010	0.000	0.011	0	0	0	0
	0.000	0.010	0.011	0	0	0	0
Sequential	1.420	8.681	126.503	0	306	0	0
Scan (With	1.150	9.171	109.603	0	306	0	0
Output	1.500	8.541	107.530	0	306	0	0
to Screen)	1.410	9.261	162.788	0	306	0	0
	1.610	9.131	144.604	0	306	0	0
	1.270	10.451	198.931	0	306	0	0
Sequential	0.010	0.040	0.050	0	295	0	0
Scan (Without	0.020	0.040	0.057	0	295	0	0
Output	0.020	0.020	0.047	0	295	0	0
to Screen)	0.020	0.030	0.047	0	295	0	0
	0.020	0.030	0.048	0	295	0	0
	0.000	0.070	0.071	0	295	0	0

Appendix B. Test Programs

B.1 Test Program: adaobj.mk (for adaobj.a)

```
include $(OS_ROOTDIR)/etc/ost>re.lib.mk
LDLIBS   = -los -losc -loscol
LIB_PATH = /tmp_mnt/home/cub2/lchou/dowork/osfilec
OS_COMPILATION_SCHEMA_DB_PATH = /lchou/test/acnote.csdb
OS_APPLICATION_SCHEMA_DB_PATH = /lchou/ter:/acnote.asdb
EXCUTABLE = adaobj
OBJECTS   = .os_schema.o adaobj.o statis.o CommandStats.o
SCHEMA_SOURCE = adaobj.cc
CPPFLAGS = -gx -I..
SCHEMA   = schema_adaobj

adaobj: $(OBJECTS)
a.make -L .. STATIS_ADA -f statis_ada.a
a.make -L ..  adaobj -f adaobj.a
mv a.out adaobj
$(OS_ROOTDIR)/lib/patch adaobj

CommandStats.o: CommandStats.c
cc $(CPPFLAGS) $(CFLAGS) -c CommandStats.c

statis.o: statis.c
cc $(CPPFLAGS) $(CFLAGS) -c statis.c

clean_obj:
osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
osrm -f $(OS_APPLICATION_SCHEMA_DB_PATH)
rm -f $(EXCUTABLE) $(OBJECTS) $(SCHEMA)

include ../ada_new.mk
```

B.2 Test Program: adaobj.a

```
with OS_TYPES; use OS_TYPES;
with OSTORE; use OSTORE;
with OSTORE_GENERICS;
with TEXT_IO; use TEXT_IO;
with LANGUAGE; use LANGUAGE;
with STATIS_ADA;

procedure ADAOBJ is
pragma LINK_WITH("-Bstatic .os_schema.o ../libosada.a adaobj.o  statis.o
   CommandStats.o -L/usr/local/objectstore/sun4/lib -los -losc");
subtype NOTE_STRING is STRING(1..20);
type LINK_NOTE;
type LINK_NOTE_PTR is access LINK_NOTE;
type LINK_NOTE is
record
priority: INTEGER;
name    : NOTE_STRING;
note    : string(1..80);
       next    : link_note_ptr;
end record;

package INT_IO is new integer_io(INTEGER);
use INT_IO;

subtype CHOICE_TYPE  is integer range 0 .. 5;
package CHOICE_IO    is new integer_io(CHOICE_TYPE);

function c_link_note_typespec return OS_TYPESPEC;
pragma INTERFACE(C, c_link_note_typespec);
pragma INTERFACE_NAME(c_link_note_typespec,
                      C_SUBP_PREFIX & "c_link_note_typespec");
package PERS_NOTE is new OSTORE_GENERICS(LINK_NOTE, LINK_NOTE_PTR,
                                         c_link_note_typespec);


ROOT     : DATABASE_ROOT;
DB       : DATABASE;
TX       : TRANSACTION;
N_IN     : INTEGER;
MYCHOICE : CHOICE_TYPE;
--
--
procedure STRCPY(NAME      : out string;
                 NOTE_NAME : in  string) is
LEN : natural := NOTE_NAME'LENGTH;
begin
  NAME(1.. LEN) := NOTE_NAME(1..LEN);
end STRCPY;
--
```

```
--
function DATABASE_RETRIEVE(NUMBER : integer;
                           HEAD   : LINK_NOTE_PTR) return LINK_NOTE_PTR is
TEMP : LINK_NOTE_PTR;
begin
  TEMP := HEAD;
     while (NUMBER /= TEMP.PRIORITY) and then (TEMP /= NULL)
        loop
           TEMP := TEMP.NEXT;
     end loop;
  return TEMP;
end DATABASE_RETRIEVE;
--
--
function INSERT(HEAD  : LINK_NOTE_PTR;
                N     : LINK_NOTE_PTR) return LINK_NOTE_PTR is
begin
       N.NEXT := HEAD;
return N;
end INSERT;

procedure DISPLAY_NOTE(N : LINK_NOTE_PTR) is
begin
  put(N.PRIORITY);
       put_line("    " & N.NAME);
       put_line(N.NOTE);
       new_line;
end DISPLAY_NOTE;
--
--
procedure TRAVERSE(HEAD  : LINK_NOTE_PTR;
                   IO    : integer) is
TEMP : LINK_NOTE_PTR;
begin
   TEMP := HEAD;
   while TEMP /= null loop
     if IO /= 0 then
         DISPLAY_NOTE(TEMP);
     end if;
     TEMP := TEMP.NEXT;
   end loop;
end TRAVERSE;
--
--
procedure DATA_SCAN is
HEAD   : LINK_NOTE_PTR;
begin
   DB := DATABASE_OPEN("/lchou/test/abnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
HEAD := PERS_NOTE.DATABASE_ROOT_GET_VALUE(ROOT);
```

```
-- start time
   STATIS_ADA.COMMANDSTATS(1);
   TRAVERSE(HEAD,1);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
   TRANSACTION_COMMIT(TX);
  DATABASE_CLOSE(DB);
end DATA_SCAN;
--
--
procedure DATA_SCAN_NIO is
HEAD   : LINK_NOTE_PTR;
begin
   DB := DATABASE_OPEN("/lchou/test/abnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
HEAD := PERS_NOTE.DATABASE_ROOT_GET_VALUE(ROOT);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
   TRAVERSE(HEAD,0);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
   TRANSACTION_COMMIT(TX);
  DATABASE_CLOSE(DB);
end DATA_SCAN_NIO;
--
--
PROCEDURE DATA_RETRIEVE is
INPUT_NUMBER : integer;
BASIC_NOTE,
HEAD         : LINK_NOTE_PTR;
begin
   put_line("Retrive a record, the priority is  5000");
   INPUT_NUMBER := 5000;
   DB := DATABASE_OPEN("/lchou/test/abnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
HEAD := PERS_NOTE.DATABASE_ROOT_GET_VALUE(ROOT);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
      BASIC_NOTE := DATABASE_RETRIEVE(INPUT_NUMBER, HEAD);
      DISPLAY_NOTE(BASIC_NOTE);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
   TRANSACTION_COMMIT(TX);
  DATABASE_CLOSE(DB);
end data_retrieve;
--
--
procedure DATABASE_OPEN_CLOSE is
HEAD         : LINK_NOTE_PTR;
```

```
COUNT        : integer;

begin
   COUNT := 1;
-- start time
   STATIS_ADA.COMMANDSTATS(1);
--    repeat opening and closing a database 10 times
  while count <= 10 loop
   DB := DATABASE_OPEN("/lchou/test/abnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
        HEAD := PERS_NOTE.DATABASE_ROOT_GET_VALUE(ROOT);
   TRANSACTION_COMMIT(TX);
   DATABASE_CLOSE(DB);
   COUNT := COUNT + 1;
   end loop;
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
end DATABASE_OPEN_CLOSE;
--
--
procedure INITIAL_DB is
BASIC_NOTE,
HEAD     : LINK_NOTE_PTR;
COUNTER  : integer := 1;
begin

   DB := DATABASE_OPEN("/lchou/test/abnote.db", FALSE, 8#664#);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
   TX := TRANSACTION_BEGIN;
       ROOT := DATABASE_ROOT_FIND("ahead", DB);
   if invalid(ROOT) then
       ROOT := DATABASE_CREATE_ROOT(DB, "ahead");
       HEAD := PERS_NOTE.DATABASE_ROOT_GET_VALUE(ROOT);
       for COUNTER in 1 .. 10000 loop
          BASIC_NOTE := PERS_NOTE.PERSISTENT_NEW(DB);
          case (COUNTER mod 10) is
          when 0 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Danile");
           STRCPY(BASIC_NOTE.NOTE ,"you need meet your friend tomorrow");

          when 1 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Susan");
           STRCPY(BASIC_NOTE.NOTE ,"you need meet Course commitee at 9:00");

          when 2 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Li");
```

```
            STRCPY(BASIC_NOTE.NOTE ,"Enjoy the silent night in Lab");

           when 3 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Chars");
            STRCPY(BASIC_NOTE.NOTE ,"You may meet me at 11:00");

           when 4 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Mike");
            STRCPY(BASIC_NOTE.NOTE ,"We have an appointment with principle");

           when 5 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Coan");
            STRCPY(BASIC_NOTE.NOTE ,"We found a book you lost");

           when 6 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Nancy");
            STRCPY(BASIC_NOTE.NOTE ,"Study Chapter 10 of OS");

           when 7 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Patric");
            STRCPY(BASIC_NOTE.NOTE ,"Please collect class addresses.");

           when 8 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Jenny");
            STRCPY(BASIC_NOTE.NOTE ,"Happy New Year");

           when 9 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Amy");
            STRCPY(BASIC_NOTE.NOTE ,"Merry Christmas");

           when others => null;
         end case;
         HEAD := INSERT(HEAD, BASIC_NOTE);
    end loop;
    PERS_NOTE.DATABASE_ROOT_SET_VALUE(ROOT, HEAD);
    else
     put_line("DATABASE abnote.db already exist !!, osrm it !!!");
    end if;
    TRANSACTION_COMMIT(TX);
    DATABASE_CLOSE(DB);
-- stop time
    STATIS_ADA.COMMANDSTATS(0);

end INITIAL_DB;
```

```
-----
begin   -- NOTE
   INIT_ADA_INTERFACE;
    loop
      loop
        begin
         put_line("**    TESTING MENU    **");
         put_line(" 0. INITIAL DATABASE : ABNOTE.DB");
         put_line(" 1. TESTING THE OPENING(CLOSING) DATABASE");
         put_line(" 2. TESTING THE LOOKUP AND RETRIEVE");
         put_line(" 3. TESTING THE SEQUENTIAL SCANING");
         put_line(" 4. TESTING THE SEQUENTIAL SCANING (WITHOUT OUTPUT)");
         put_line(" 5. BYE !!");
         PUT("INPUT -> ");
         choice_io.get(MYCHOICE);
         text_io.skip_line;
        exit;
         exception
            when data_error | constraint_error =>
                text_io.skip_line;
                text_io.put_line("Your choice must be between 0 and 5");
                text_io.new_line;
       end;
      end loop;
  --
  -- do different tasks from here
  --
    case MYCHOICE is
        when 0 =>
           INITIAL_DB;
        when 1 =>
           DATABASE_OPEN_CLOSE;
        when 2 =>
           DATA_RETRIEVE;
        when 3 =>
           DATA_SCAN;
        when 4 =>
           DATA_SCAN_NIO;
        when 5 =>
           exit;
        when others => null;
    end case;
  end loop;
end ADAOBJ;
```

B.3 Test Program: adacol.mk (for adacol.a)

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk
LDLIBS   = -los -losc -loscol
LIB_PATH = /tmp_mnt/home/cub2/lchou/dowork/osfilec
OS_COMPILATION_SCHEMA_DB_PATH = /lchou/test/acnote.csdb
OS_APPLICATION_SCHEMA_DB_PATH = /lchou/test/acnote.asdb
EXCUTABLE = adacol
OBJECTS   = .os_schema.o adacol.o statis.o CommandStats.o
SCHEMA_SOURCE = adacol.cc
CPPFLAGS = -gx -I..
SCHEMA  = schema_adacol

adacol: $(OBJECTS)
a.make -L .. STATIS_ADA -f statis_ada.a
a.make -L ..  adacol -f adacol.a
mv a.out adacol
$(OS_ROOTDIR)/lib/patch adacol

CommandStats.o: CommandStats.c
cc $(CPPFLAGS) $(CFLAGS) -c CommandStats.c

statis.o: statis.c
cc $(CPPFLAGS) $(CFLAGS) -c statis.c

clean_col:
osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
osrm -f $(OS_APPLICATION_SCHEMA_DB_PATH)
rm -f $(EXCUTABLE) $(OBJECTS) $(SCHEMA)


include ../ada_new.mk
```

B.4 Test Program: adacol.a

```
with OS_TYPES; use OS_TYPES;
with OSTORE; use OSTORE;
with OSTORE_GENERICS;
with OS_COLLECTION_PKG;
with OS_CURSOR_PKG;
with TEXT_IO; use TEXT_IO;
with LANGUAGE; use LANGUAGE;
with STATIS_ADA;

procedure ADACOL is
pragma LINK_WITH("-Bstatic .os_schema.o ../libosada.a adacol.o statis.o
   CommandStats.o -L/usr/local/objectstore/sun4/lib -los -losc -loscol");
subtype NOTE_STRING is STRING(1..20);
type NOTE_COL;
type NOTE_COL_PTR is access NOTE_COL;
type NOTE_COL is
record
priority: INTEGER;
name    : NOTE_STRING;
note    : string(1..80);
       next    : note_col_ptr;
end record;

package INT_IO is new integer_io(INTEGER);
use INT_IO;

subtype CHOICE_TYPE  is integer range 0 .. 5;
package CHOICE_IO    is new integer_io(CHOICE_TYPE);

function c_note_col_typespec return OS_TYPESPEC;
pragma INTERFACE(C, c_note_col_typespec);
pragma INTERFACE_NAME(c_note_col_typespec,
                      C_SUBP_PREFIX & "c_note_col_typespec");
package PERS_NOTE is new OSTORE_GENERICS(NOTE_COL, NOTE_COL_PTR,
                                         c_note_col_typespec);

--
-- os_collection's type
function c_os_collection_typespec return OS_TYPESPEC;
pragma INTERFACE(C, c_os_collection_typespec);
pragma INTERFACE_NAME(c_os_collection_typespec,
                      C_SUBP_PREFIX & "c_os_collection_typespec");


package COLL_NOTES is new OSTORE_GENERICS(U_TYPE     => NOTE_COL,
                                          U_TYPEPTR => NOTE_COL_PTR,
                                  GET_OS_TYPESPEC => c_os_collection_typespec);
```

```
package OS_COLL  is new OS_COLLECTION_PKG(U_TYPE => NOTE_COL,
                                          U_TYPEPTR => NOTE_COL_PTR,
                                       GET_OS_TYPESPEC =>c_os_collection_typespec);

package OS_CURSORS  is new OS_CURSOR_PKG(U_TYPE => NOTE_COL,
                                          U_TYPEPTR => NOTE_COL_PTR);



ROOT     : DATABASE_ROOT;
DB       : DATABASE;
TX       : TRANSACTION;
N_IN     : INTEGER;
MYCHOICE : CHOICE_TYPE;
--
--
procedure STRCPY(NAME      : out string;
                 NOTE_NAME : in  string) is
LEN : natural := NOTE_NAME'LENGTH;
begin
  NAME(1.. LEN) := NOTE_NAME(1..LEN);
end STRCPY;
--
--
procedure DISPLAY_NOTE(N : NOTE_COL_PTR) is
begin
  put(N.PRIORITY);
       put_line("    " & N.NAME);
       put_line(N.NOTE);
       new_line;
end DISPLAY_NOTE;
--
--
procedure DATA_SCAN is
HEAD   : OS_COLLECTION;
P            : NOTE_COL_PTR;
CUR          : OS_CURSOR;

begin
   DB := DATABASE_OPEN("/lchou/test/acnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
HEAD := COLL_NOTES.DATABASE_ROOT_GET_VALUE(ROOT);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
-- iteration
         CUR :=  OS_CURSORS.OS_CURSOR_CREATE(HEAD);
         P := NOTE_COL_PTR(OS_CURSORS.OS_CURSOR_FIRST(CUR));
         while (OS_CURSORS.OS_CURSOR_MORE(CUR)) loop
              DISPLAY_NOTE(P);
              P:=OS_CURSORS.OS_CURSOR_NEXT(CUR);
```

```
        end loop;
        OS_CURSORS.OS_CURSOR_DELETE(CUR);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
   TRANSACTION_COMMIT(TX);
  DATABASE_CLOSE(DB);
end DATA_SCAN;
--
--
procedure DATA_SCAN_NIO is
HEAD   : OS_COLLECTION;
P            : NOTE_COL_PTR;
CUR          : OS_CURSOR;

begin
   DB := DATABASE_OPEN("/lchou/test/acnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
HEAD := COLL_NOTES.DATABASE_ROOT_GET_VALUE(ROOT);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
-- iteration
        CUR :=  OS_CURSORS.OS_CURSOR_CREATE(HEAD);
        P := NOTE_COL_PTR(OS_CURSORS.OS_CURSOR_FIRST(CUR));
        while (OS_CURSORS.OS_CURSOR_MORE(CUR)) loop
             P:=OS_CURSORS.OS_CURSOR_NEXT(CUR);
        end loop;
        OS_CURSORS.OS_CURSOR_DELETE(CUR);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
   TRANSACTION_COMMIT(TX);
  DATABASE_CLOSE(DB);
end DATA_SCAN_NIO;
--
--

PROCEDURE DATA_RETRIEVE is
INPUT_STRING : STRING(1 .. 130) :=(others => ' ');
QUERIED_NOTE,
HEAD         : OS_COLLECTION;
P            : NOTE_COL_PTR;
CUR          : OS_CURSOR;

begin
   put_line("Retrive a record, the priority is  5000");
   DB := DATABASE_OPEN("/lchou/test/acnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
HEAD := COLL_NOTES.DATABASE_ROOT_GET_VALUE(ROOT);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
```

```
      STRCPY(INPUT_STRING, "priority == 5000");
      QUERIED_NOTE := OS_COLL.OS_COLLECTION_QUERY(HEAD,"note_col*"
                  ,INPUT_STRING,DB);
-- iteration
        CUR :=  OS_CURSORS.OS_CURSOR_CREATE(QUERIED_NOTE);
        P := NOTE_COL_PTR(OS_CURSORS.OS_CURSOR_FIRST(CUR));
        while (OS_CURSORS.OS_CURSOR_MORE(CUR)) loop
             DISPLAY_NOTE(P);
             P:=OS_CURSORS.OS_CURSOR_NEXT(CUR);
        end loop;
        OS_CURSORS.OS_CURSOR_DELETE(CUR);
 -- stop time
   STATIS_ADA.COMMANDSTATS(0);
   TRANSACTION_COMMIT(TX);
  DATABASE_CLOSE(DB);
end data_retrieve;
--
--
procedure DATABASE_OPEN_CLOSE is
HEAD         : OS_COLLECTION;
COUNT        : integer;

begin
   COUNT := 1;
-- start time
   STATIS_ADA.COMMANDSTATS(1);
--    repeat opening and closing a database 10 times
  while count <= 10 loop
   DB := DATABASE_OPEN("/lchou/test/acnote.db", FALSE, 8#664#);
   TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("ahead", DB);
        HEAD := COLL_NOTES.DATABASE_ROOT_GET_VALUE(ROOT);
   TRANSACTION_COMMIT(TX);
   DATABASE_CLOSE(DB);
   COUNT := COUNT + 1;
   end loop;
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
end DATABASE_OPEN_CLOSE;
--
--
procedure INITIAL_DB is
BASIC_NOTE : NOTE_COL_PTR;
HEAD       : OS_COLLECTION;
COUNTER  : integer := 1;
begin

   DB := DATABASE_OPEN("/lchou/test/acnote.db", FALSE, 8#664#);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
   TX := TRANSACTION_BEGIN;
```

```
    ROOT := DATABASE_ROOT_FIND("ahead", DB);
if invalid(ROOT) then
    ROOT := DATABASE_CREATE_ROOT(DB, "ahead");
    HEAD := OS_COLL.OS_COLLECTION_CREATE(DB,
        "maintain_cursors | maintain_order",10);
    for COUNTER in 1 .. 10000 loop
       BASIC_NOTE := PERS_NOTE.PERSISTENT_NEW(DB);
       case (COUNTER mod 10) is
       when 0 =>
        BASIC_NOTE.PRIORITY := counter ·
        STRCPY(BASIC_NOTE.NAME ,"Danile");
        STRCPY(BASIC_NOTE.NOTE ,"you need meet your friend tomorrow");

       when 1 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Susan");
        STRCPY(BASIC_NOTE.NOTE ,"you need meet Course commitee at 9:00");

       when 2 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Li");
        STRCPY(BASIC_NOTE.NOTE ,"Enjoy the silent night in Lab");

       when 3 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Chars");
        STRCPY(BASIC_NOTE.NOTE ,"You may meet me at 11:00");

       when 4 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Mike");
        STRCPY(BASIC_NOTE.NOTE ,"We have an appointment with principle");

       when 5 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Coan");
        STRCPY(BASIC_NOTE.NOTE ,"We found a book you lost");

       when 6 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Nancy");
        STRCPY(BASIC_NOTE.NOTE ,"Study Chapter 10 of OS");

       when 7 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Patric");
        STRCPY(BASIC_NOTE.NOTE ,"Please collect class addresses.");

       when 8 =>
        BASIC_NOTE.PRIORITY := counter ;
        STRCPY(BASIC_NOTE.NAME ,"Jenny");
```

```
            STRCPY(BASIC_NOTE.NOTE ,"Happy New Year");

           when 9 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Amy");
            STRCPY(BASIC_NOTE.NOTE ,"Merry Christmas");

          when others => null;
        end case;
        OS_COLL.OS_COLLECTION_INSERT(HEAD, BASIC_NOTE);
    end loop;
    COLL_NOTES.DATABASE_ROOT_SET_VALUE(ROOT, HEAD);
    else
     put_line("DATABASE acnote.db already exist !!, osrm it !!!");
    end if;
    TRANSACTION_COMMIT(TX);
    DATABASE_CLOSE(DB);
-- stop time
    STATIS_ADA.COMMANDSTATS(0);

end INITIAL_DB;
-----
begin   -- NOTE
    INIT_ADA_INTERFACE;
    OS_COLL.OS_COLLECTION_INITIALIZE;
      loop
       loop
         begin
          put_line("**    TESTING MENU    **");
          put_line(" 0. INITIAL DATABASE : ACNOTE.DB");
          put_line(" 1. TESTING THE OPENING(CLOSING) DATABASE");
          put_line(" 2. TESTING THE LOOKUP AND RETRIEVE");
          put_line(" 3. TESTING THE SEQUENTIAL SCANING");
          put_line(" 4. TESTING THE SEQUENTIAL SCANING (WITHOUT OUTPUT)");
          put_line(" 5. BYE !!");
          PUT("INPUT -> ");
          choice_io.get(MYCHOICE);
          text_io.skip_line;
         exit;
          exception
             when data_error | constraint_error =>
                 text_io.skip_line;
                 text_io.put_line("Your choice must be between 0 and 5");
                 text_io.new_line;
       end;
      end loop;
 --
 -- do different tasks from here
 --
   case MYCHOICE is
       when 0 =>
```

```
            INITIAL_DB;
         when 1 =>
            DATABASE_OPEN_CLOSE;
         when 2 =>
            DATA_RETRIEVE;
         when 3 =>
            DATA_SCAN;
         when 4 =>
            DATA_SCAN_NIO;
         when 5 =>
            exit;
         when others => null;
      end case;
   end loop;
end ADACOL;
```

B.5 Test Program: adaobj.mk (for adaobj.c)

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk
OS_COMPILATION_SCHEMA_DB_PATH= /$(USER)/test/bnote.csdb
OS_APPLICATION_SCHEMA_DB_PATH= /$(USER)/test/bnote.asdb
LDLIBS = -los -losc
SOURCES =   adaobj.c  CommandStats.c skm_adao.cc  ct_bnote.c
OBJECTS =   adaobj.o  CommandStats.o skm_adao.o   ct_bnote.o
EXECUTABLES =   adaobj  ct_bnote
CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -gx
CC = cc
LIB_PATH = -L/usr/local/objectstore/sun4/lib

all: $(EXECUTABLES)

adaobj: adaobj.o  CommandStats.o  schema_standin_Adao
$(OS_PRELINK) .os_schema.cc \
    $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
    adaobj.o $(LDLIBS)
OSCC -c .os_schema.cc
$(LINK.c) -o adaobj -Bstatic adaobj.o   \
          CommandStats.o .os_schema.o $(LDLIBS)
$(OS_POSTLINK) adaobj

ct_bnote: ct_bnote.o schema_standin_Adao
$(OS_PRELINK) .os_schema.cc \
    $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
    ct_bnote.o $(LDLIBS)
OSCC -c .os_schema.cc
$(LINK.c) -o ct_bnote ct_bnote.o   .os_schema.o $(LDLIBS)
$(OS_POSTLINK) ct_bnote

adaobj.o: adaobj.c
$(CC) $(CPPFLAGS) $(CFLAGS) -c adaobj.c

CommandStats.o: CommandStats.c
$(CC) $(CPPFLAGS) $(CFLAGS) -c CommandStats.c

schema_standin_Adao: skm_adao.cc
OSCC $(CPPFLAGS) -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) skm_adao.cc
touch schema_standin_Adao

clean:
osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
osrm -f $(OS_APPLICATION_SCHEMA_DB_PATH)
rm -f $(EXECUTABLES) $(OBJECTS) schema_standin_Adao

depend: .depend_B

.depend_B:
osmakedep .depend_B $(CPPFLAGS) -files $(SOURCES)
```

```
include .depend_B
```

B.6 Test Program: adaobj.c

```
/* file :  adaobj.c program - main file
           ObjectStore  C library
   implemented by Li Chou, in Jan 1993.
*/
#include <stdio.h>
#include <ostore/ostore.h>
#include <strings.h>
#include <sys/time.h>
#include <sys/resource.h>

#include "adaobj.h"


extern FILE *basic_file;
void initial_db();
void database_open_close();
void data_retrieve();
void data_scan();
void data_scan_nio();
struct link_note *database_retrieve();
void display_note();
void traverse();

void exit();

      /* global timing variables for CommandStats -- Jacobs 18/09/91 */
    static struct timeval elapsed;
    static struct rusage  exec;
 database *db;
 database_root *root;
    /* allocat os_typespec   */
 os_typespec *note_pad_type;

main()
{
  char     choose;

  start_objectstore();
  note_pad_type = alloc_typespec("link_note",0);

  /*-- main loop --
    drive manual
  -- choose input by user   */
 while (1) {
   printf("**    Testing Menu    **\n");
   printf(" 0. Initial database : bnote.db\n");
   printf(" 1. Testing the opening(closing) database\n");
   printf(" 2. Testing the lookup and retrieve\n");
   printf(" 3. Testing the sequential scaning\n");
```

```
    printf(" 4. Testing the sequential scaning (without output to screen)\n");
    printf(" 5. Bye !!\n");
    printf("Input -> ");
   while (scanf("%c" , &choose) ==1) {
      if (choose <= '5' && choose >= '0')
           break;
       }
    if (choose == '5') exit(1);

    switch (choose) {
      case  '0' :
         initial_db();
         break;
      case  '1' :
         database_open_close();
         break;
      case  '2' :
         data_retrieve();
         break;
      case  '3' :
         data_scan();
         break;
      case  '4' :
         data_scan_nio();
         break;
     }
   }
}

void database_open_close()
 {
struct link_note *head;
int count;

  count = 1;
  /* Start time commandstat  */
  CommandStats(1, stdout, &elapsed, &exec);
  /* repeat opening and closing a database 10 times  */
  while (count <= 10) {
   db = database_lookup_open("/lchou/test/bnote.db", 0, 0664);
   OS_BEGIN_TXN(tx1,0,transaction_update) {
     root = database_root_find("ahead", db);
     head = database_root_get_value(root, note_pad_type);
   } OS_END_TXN(tx1);
   database_close(db);
   ++count;
   }
  /* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
}
```

```
void data_retrieve()
 {

int  input_number;
struct link_note *basic_note, *head;


printf("Retrive a record, the priority is  5000\n");
/*  if (scanf("%d", &input_number) == 1) */
   input_number = 5000;
  db = database_lookup_open("/lchou/test/bnote.db", 0, 0664);
  OS_BEGIN_TXN(tx1,0,transaction_update) {
     root = database_root_find("ahead", db);
     head = database_root_get_value(root, note_pad_type);
/* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     basic_note = database_retrieve(input_number, head);
       display_note(basic_note);
/* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
      } OS_END_TXN(tx1);
 database_close(db);
}

void data_scan() {
struct link_note *head;

     db = database_lookup_open("/lchou/test/bnote.db", 0, 0664);
OS_BEGIN_TXN(tx1,0,transaction_update) {
     root = database_root_find("ahead", db);
     head = database_root_get_value(root, note_pad_type);
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     traverse(head,1);
    /* Stop time commandstat */
     CommandStats(0, stdout, &elapsed, &exec);
  } OS_END_TXN(tx1);
database_close(db);
}

void data_scan_nio() {
struct link_note *head;
void traverse();

     db = database_lookup_open("/lchou/test/bnote.db", 0, 0664);
OS_BEGIN_TXN(tx1,0,transaction_update) {
     root = database_root_find("ahead", db);
     head = database_root_get_value(root, note_pad_type);
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     traverse(head,0);
```

```
    /* Stop time commandstat */
     CommandStats(0, stdout, &elapsed, &exec);
  } OS_END_TXN(tx1);
database_close(db);}



/* retrieve a note  */
struct link_note *database_retrieve(numb, link_head)
int numb;
struct link_note *link_head;
{
struct link_note *temp;
temp = link_head;

 while ((numb != temp -> priority ) && (temp != NULL)){
    temp = temp-> next;
  }
 return temp;
 }


/* Print out to the specified stream this note */
void display_note(n)
struct link_note *n;
{
      printf("priority (%d)      name %s \n", n->priority, n->name);
      printf("note  %s\n", n -> note);
}

/* Insert a node   */
struct link_note *insert(p, q)
struct link_note *p;
struct link_note *q;
{
   q -> next= p;
   return q;
}

/* Sequencial scaning */
void traverse(anote,io)
NOTE anote;
int io;
{
NOTE temp;

 temp = anote;
 while (temp != NULL) {
   if (io != 0) {
       display_note(temp);
```

```
    }
  temp = temp -> next;
 }
}

void initial_db()
 {
    /* global timing variables for CommandStats -- Jacobs 18/09/91 */
   static struct timeval elapsed;
   static struct rusage  exec;
   database *db;
   database_root *root;
   struct link_note  *head, *basic_note;
   int     counter;

  db = database_lookup_open("/lchou/test/bnote.db", 0, 0664);
     /* Start time commandstat  */
  CommandStats(1, stdout, &elapsed, &exec);
  OS_BEGIN_TXN(tx1,0,transaction_update) {
    root = database_root_find("ahead", db);
    if (!root) {
       root = database_create_root(db, "ahead");
    }
    else
    {
      printf("The database exist !!   osrm it first !!!\n");
      exit();
    }
    head = database_root_get_value(root, note_pad_type);
    for (counter = 1; counter <= 10000; ++counter){
      basic_note = (struct link_note*) objectstore_alloc(note_pad_type, 1, db);
       switch (counter - ((int)(counter/10)) * 10) {
         case 0 :
          basic_note -> priority = counter ;
          strcpy(basic_note -> name ,"Danile");
          strcpy(basic_note -> note ,"you need meet your friend tomorrow");
          break;
         case 1 :
          basic_note -> priority = counter ;
          strcpy(basic_note -> name ,"Susan");
          strcpy(basic_note -> note ,"you need meet Course commitee at 9:00");
          break;

         case 2:
          basic_note -> priority = counter ;
          strcpy(basic_note -> name ,"Li");
          strcpy(basic_note -> note ,"Enjoy the silent night in Lab");
          break;

         case 3 :
          basic_note -> priority = counter ;
```

```
            strcpy(basic_note -> name ,"Chars");
            strcpy(basic_note -> note ,"You may meet me at 11:00");
            break;
           case 4 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Mike");
            strcpy(basic_note -> note ,"We have an appointment with principle");
            break;
           case 5 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Coan");
            strcpy(basic_note -> note ,"We found a book you lost");
            break;
           case 6 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Nancy");
            strcpy(basic_note -> note ,"Study Chapter 10 of OS");
            break;
           case 7 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Patric");
            strcpy(basic_note -> note ,"Please collect class addresses.");
            break;
           case 8 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Jenny");
            strcpy(basic_note -> note ,"Happy New Year");
            break;
           case 9 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Amy");
            strcpy(basic_note -> note ,"Merry Christmas");
            break;
              }
   head = insert(head, basic_note);

   }
  database_root_set_value(root, head, note_pad_type);
  } OS_END_TXN(tx1);
  database_close(db);
/* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
}
```

B.7 Test Program: adacol.mk (for adacol.c)

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk
OS_COMPILATION_SCHEMA_DB_PATH= /$(USER)/test/cnote.cndb
OS_APPLICATION_SCHEMA_DB_PATH= /$(USER)/test/cnote.asdb
LDLIBS = -los -losc -loscol
SOURCES =   adacol.c  CommandStats.c  skm_adac.cc  ct_cnote.c
OBJECTS =   adacol.o  CommandStats.o  skm_adac.o   ct_cntoe.o
EXECUTABLES =   adacol  ct_cnote
CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -gx
CC = cc
LIB_PATH = -L/usr/local/objectstore/sun4/lib

all: $(EXECUTABLES)

adacol: adacol.o  CommandStats.o   schema_standin_Adacol
$(OS_PRELINK) .os_schema.cc \
    $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
    adacol.o $(LDLIBS)
OSCC -c .os_schema.cc
$(LINK.c) -o adacol -Bstatic adacol.o CommandStats.o .os_schema.o \
          $(LDLIBS)
$(OS_POSTLINK) adacol

ct_cnote: ct_cnote.o  schema_standin_Adacol
$(OS_PRELINK) .os_schema.cc \
    $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
    ct_cnote.o $(LDLIBS)
OSCC -c .os_schema.cc
$(LINK.c) -o ct_cnote ct_cnote.o .os_schema.o $(LDLIBS)
$(OS_POSTLINK) ct_cnote

ct_cnote.o: ct_cnote.c
$(CC) $(CPPFLAGS) $(CFLAGS) -c ct_cnote.c

adacol.o: adacol.c
$(CC) $(CPPFLAGS) $(CFLAGS) -c adacol.c

CommandStats.o: CommandStats.c
$(CC) $(CPPFLAGS) $(CFLAGS) -c CommandStats.c

schema_standin_Adacol: skm_adac.cc
OSCC $(CPPFLAGS) -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH)  \
        skm_adac.cc
touch schema_standin_Adacol

clean:
osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
osrm -f $(OS_APPLICATION_SCHEMA_DB_PATH)
rm -f $(EXECUTABLES) $(OBJECTS) schema_standin_Adacol
```

```
depend: .depend_C

.depend_C:
osmakedep .depend_C $(CPPFLAGS) -files $(SOURCES)

include .depend_C
```

B.8 Test Program: adacol.c

```
/* file :  adacol.c program - main file

          ObjectStore  C library - collection functions
   implemented by Li Chou, in Jan 1993.
*/

#include <stdio.h>
#include <ostore/ostore.h>
#include <ostore/coll.h>
#include <strings.h>
#include <sys/time.h>
#include <sys/resource.h>

#include "adacol.h"


 extern FILE *basic_file;
 void initial_col();
 void database_open_close();
 void data_retrieve();
 void data_scan();
 void data_scan_nio();
 void display_note();

 void exit();

   /* global timing variables for CommandStats -- Jacobs 18/09/91 */
 static struct timeval elapsed;
 static struct rusage  exec;
  database *db;
  database_root *root;
  os_typespec *note_type, *os_coll_type;

 main()
{
  char     choose;

  start_objectstore();
  os_collection_initialize();
  /* allocat os_typespec   */
    note_type    = alloc_typespec("note_col",0);
    os_coll_type = alloc_typespec("os_collection",0);
  /*-- main loop --
    drive manual
  -- choose input by user   */

while (1) {
   printf("**    Testing Menu    **\n");
   printf(" 0. Initial database : cnote.db\n");
   printf(" 1. Testing the opening(closing) database\n");
```

```
    printf(" 2. Testing the lookup and retrieve\n");
    printf(" 3. Testing the sequential scaning\n");
    printf(" 4. Testing the sequential scaning (without output to screen)\n");
    printf(" 5. Bye !!\n");
    printf("Input -> ");
   while (scanf("%c" , &choose) ==1) {
      if (choose <= '5' && choose >= '0')
            break;
       }
    if (choose == '5') exit(1);

    switch (choose) {
      case  '0' :
         initial_col();
         break;
      case  '1' :
         database_open_close();
         break;
      case  '2' :
         data_retrieve();
         break;
      case  '3' :
         data_scan();
         break;
      case  '4'
         data_scan_nio();
         break;

     }
   }
}

void database_open_close()
 {
os_collection *head;
int count;

  count = 1;
  /* Start time commandstat  */
  CommandStats(1, stdout, &elapsed, &exec);
  /* repeat opening and closing a database 10 times  */
  while (count <= 10) {
   db = database_lookup_open("/lchou/test/cnote.db", 0, 0664);
   OS_BEGIN_TXN(tx1,0,transaction_update) {
      root =  database_root_find("ahead", db);
      head = (os_collection*) database_root_get_value(root, os_coll_type);
    } OS_END_TXN(tx1);
   database_close(db);
   ++count;
   }
  /* Stop time commandstat */
```

```
   CommandStats(0, stdout, &elapsed, &exec);
}

void data_retrieve()
 {
os_collection *head, *queried_note;
os_cursor  *cur;
struct note_col  *p;
char *input_string;

printf("Retrive a record, the priority is  5000\n");
/*  if (scanf("%d", &input_number) == 1) */

  db = database_lookup_open("/lchou/test/cnote.db", 0, 0664);
  OS_BEGIN_TXN(tx1,0,transaction_update) {
     root = database_root_find("ahead", db);
     head = (os_collection*) database_root_get_value(root, os_coll_type);
/* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
      strcpy(input_string,"priority == 5000");
      queried_note = os_collection_query(head, "note_col*", input_string,
                     db,0 ,0);
      cur =   os_cursor_create(queried_note, 0);
        for (p = (struct note_col*) os_cursor_first(cur); os_cursor_more(cur);
         p =  (struct note_col*) os_cursor_next(cur))
         display_note(p) ;

        os_cursor_delete(cur);
/* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
      } OS_END_TXN(tx1);
 database_close(db);
}

void data_scan()
{
os_collection *head;
os_cursor  *cur;
struct note_col  *p;

     db = database_lookup_open("/lchou/test/cnote.db", 0, 0664);
OS_BEGIN_TXN(tx1,0,transaction_update) {
     root =  database_root_find("ahead", db);
     head = (os_collection*) database_root_get_value(root, os_coll_type);
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     cur =   os_cursor_create(head, 0);
       for (p = (struct note_col*) os_cursor_first(cur); os_cursor_more(cur);
         p =  (struct note_col*) os_cursor_next(cur))
         display_note(p) ;
```

```
     os_cursor_delete(cur);
    /* Stop time commandstat */
     CommandStats(0, stdout, &elapsed, &exec);
  } OS_END_TXN(tx1);
database_close(db);
}

void data_scan_nio()
{
os_collection *head;
os_cursor  *cur;
struct note_col  *p;

     db = database_lookup_open("/lchou/test/cnote.db", 0, 0664);
OS_BEGIN_TXN(tx1,0,transaction_update) {
     root =  database_root_find("ahead", db);
     head = (os_collection*) database_root_get_value(root, os_coll_type);
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     cur =   os_cursor_create(head, 0);
       for (p = (struct note_col*) os_cursor_first(cur); os_cursor_more(cur);
         p =  (struct note_col*) os_cursor_next(cur));

        os_cursor_delete(cur);
    /* Stop time commandstat */
     CommandStats(0, stdout, &elapsed, &exec);
  } OS_END_TXN(tx1);
database_close(db);
}

/* Print out to the specified stream this note */
void display_note(n)
struct note_col *n;
{
      printf("priority (%d)      name %s \n", n->priority, n->name);
      printf("note  %s\n", n -> note);
}

void initial_col() {

  struct note_col  *head, *basic_note;
  int     counter;

   db = database_lookup_open("/lchou/test/cnote.db", 0, 0664);
/* Start time commandstat  */
   CommandStats(1, stdout, &elapsed, &exec);
   OS_BEGIN_TXN(tx1,0,transaction_update) {
     root =  database_root_find("ahead", db);
     if (!root) {
        root =  database_create_root(db, "ahead");
        head = os_collection_create(db,
```

```
                os_collection_maintain_cursors |
                os_collection_maintain_order,10,0,0);
    database_root_set_value(root, head, os_coll_type);
}
else
{
 printf("The database exist !!   osrm it first !!!\n");
  exit();
}

head = database_root_get_value(root, os_coll_type);
for (counter = 1; counter <= 10000; ++counter){
  basic_note = (struct note_col*) objectstore_allcc(note_type, 1, db);
   switch (counter - ((int)(counter/10)) * 10) {
     case 0 :
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Danile");
      strcpy(basic_note -> note ,"you need meet your friend tomorrow");
      break;

     case 1 :
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Susan");
      strcpy(basic_note -> note ,"you need meet Course commitee at 9:00");
      break;

     case 2:
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Li");
      strcpy(basic_note -> note ,"Enjoy the silent night in Lab");
      break;

     case 3 :
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Chars");
      strcpy(basic_note -> note ,"You may meet me at 11:00");
      break;

     case 4 :
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Mike");
      strcpy(basic_note -> note ,"We have an appointment with principle");
      break;
     case 5 :
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Coan");
      strcpy(basic_note -> note ,"We found a book you lost");
      break;
     case 6 :
      basic_note -> priority = counter ;
      strcpy(basic_note -> name ,"Nancy");
```

```
            strcpy(basic_note -> note ,"Study Chapter 10 of OS");
            break;
           case 7 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Patric");
            strcpy(basic_note -> note ,"Please collect class addresses.");
            break;
           case 8 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Jenny");
            strcpy(basic_note -> note ,"Happy New Year");
            break;
           case 9 :
            basic_note -> priority = counter ;
            strcpy(basic_note -> name ,"Amy");
            strcpy(basic_note -> note ,"Merry Christmas");
            break;
              }
       os_collection_insert(head, basic_note);
    }
  } OS_END_TXN(txn);
  database_close(db);
/* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
}
```

B.9 Test Program: purobj.a

```
-- a pure Ada program which without accessing DBMS
with TEXT_IO; use TEXT_IO;
with STATIS_ADA;

procedure PUROBJ is
pragma LINK_WITH("-Bstatic  statis.o CommandStats.o");
subtype NOTE_STRING is STRING(1..20);
type LINK_NOTE;
type LINK_NOTE_PTR is access LINK_NOTE;
type LINK_NOTE is
record
priority: INTEGER;
name     : NOTE_STRING;
note     : string(1..80);
        next     : link_note_ptr;
end record;

package INT_IO is new integer_io(INTEGER);
use INT_IO;

subtype CHOICE_TYPE  is integer range 0 .. 5;
package CHOICE_IO    is new integer_io(CHOICE_TYPE);

HEAD     : LINK_NOTE_PTR;
N_IN     : INTEGER;
MYCHOICE : CHOICE_TYPE;
--
--
procedure STRCPY(NAME      : out string;
                 NOTE_NAME : in  string) is
LEN : natural := NOTE_NAME'LENGTH;
begin
  NAME(1.. LEN) := NOTE_NAME(1..LEN);
end STRCPY;
--
--
function DATABASE_RETRIEVE(NUMBER : integer;
                           HEAD   : LINK_NOTE_PTR) return LINK_NOTE_PTR is
TEMP : LINK_NOTE_PTR;
begin
  TEMP := HEAD;
     while (NUMBER /= TEMP.PRIORITY) and then (TEMP /= NULL)
        loop
           TEMP := TEMP.NEXT;
     end loop;
  return TEMP;
end DATABASE_RETRIEVE;
--
--
```

```
function INSERT(HEAD  : LINK_NOTE_PTR;
                N     : LINK_NOTE_PTR) return LINK_NOTE_PTR is
begin
        N.NEXT := HEAD;
return N;
end INSERT;

procedure DISPLAY_NOTE(N : LINK_NOTE_PTR) is
begin
  put(N.PRIORITY);
        put_line("    " & N.NAME);
        put_line(N.NOTE);
        new_line;
end DISPLAY_NOTE;
--
--
procedure TRAVERSE(HEAD  : LINK_NOTE_PTR;
                   IO    : integer) is

TEMP : LINK_NOTE_PTR;
begin
   TEMP := HEAD;
   while TEMP /= null loop
     if   /=0 then
         DISPLAY_NOTE(TEMP);
     end if;
     TEMP := TEMP.NEXT;
   end loop;
end TRAVERSE;
--
--
procedure DATA_SCAN_IO(HEAD : LINK_NOTE_PTR) is

begin
-- start time
   STATIS_ADA.COMMANDSTATS(1);
   TRAVERSE(HEAD,1);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
end DATA_SCAN_IO;
--
--
procedure DATA_SCAN(HEAD : LINK_NOTE_PTR) is

begin
-- start time
   STATIS_ADA.COMMANDSTATS(1);
   TRAVERSE(HEAD,0);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
end DATA_SCAN;
```

```
--
--
PROCEDURE DATA_RETRIEVE(TEMP :  LINK_NOTE_PTR) is
INPUT_NUMBER : integer;
BASIC_NOTE   : LINK_NOTE_PTR;
begin
   put_line("Retrive a record, the priority is  5000");
   INPUT_NUMBER := 5000;
-- start time
   STATIS_ADA.COMMANDSTATS(1);
      BASIC_NOTE := DATABASE_RETRIEVE(INPUT_NUMBER, TEMP);
      DISPLAY_NOTE(BASIC_NOTE);
-- stop time
   STATIS_ADA.COMMANDSTATS(0);
end data_retrieve;
--
--
function INITIAL_DB return LINK_NOTE_PTR is

HEAD,
BASIC_NOTE : LINK_NOTE_PTR;
COUNTER    : integer := 1;
begin

-- start time
   STATIS_ADA.COMMANDSTATS(1);
       for COUNTER in 1 .. 10000  loop
          BASIC_NOTE := new LINK_NOTE;
          case (COUNTER mod 10) is
          when 0 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Danile");
           STRCPY(BASIC_NOTE.NOTE ,"you need meet your friend tomorrow");

          when 1 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Susan");
           STRCPY(BASIC_NOTE.NOTE ,"you need meet Course commitee at 9:00");

          when 2 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Li");
           STRCPY(BASIC_NOTE.NOTE ,"Enjoy the silent night in Lab");

          when 3 =>
           BASIC_NOTE.PRIORITY := counter ;
           STRCPY(BASIC_NOTE.NAME ,"Chars");
           STRCPY(BASIC_NOTE.NOTE ,"You may meet me at 11:00");

          when 4 =>
           BASIC_NOTE.PRIORITY := counter ;
```

```
            STRCPY(BASIC_NOTE.NAME ,"Mike");
            STRCPY(BASIC_NOTE.NOTE ,"We have an appointment with principle");

           when 5 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Coan");
            STRCPY(BASIC_NOTE.NOTE ,"We found a book you lost");

           when 6 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Nancy");
            STRCPY(BASIC_NOTE.NOTE ,"Study Chapter 10 of OS");

           when 7 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Patric");
            STRCPY(BASIC_NOTE.NOTE ,"Please collect class addresses.");

           when 8 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Jenny");
            STRCPY(BASIC_NOTE.NOTE ,"Happy New Year");

           when 9 =>
            BASIC_NOTE.PRIORITY := counter ;
            STRCPY(BASIC_NOTE.NAME ,"Amy");
            STRCPY(BASIC_NOTE.NOTE ,"Merry Christmas");

           when others => null;
         end case;
         HEAD := INSERT(HEAD, BASIC_NOTE);
    end loop;
-- stop time
    STATIS_ADA.COMMANDSTATS(0);
   return HEAD;
end INITIAL_DB;
-----
begin   -- NOTE
     loop
       loop
         begin
          put_line("**    TESTING MENU    **");
          put_line(" 0. INITIAL DATABASE; TRANSIENT ONLY");
          put_line(" 1. TESTING THE LOOKUP AND RETRIEVE");
          put_line(" 2. TESTING THE SEQUENTIAL SCANING W I/O  ");
          put_line(" 3. TESTING THE SEQUENTIAL SCANING WITHOUT I/O");
          put_line(" 4. BYE !!");
          PUT("INPUT -> ");
          choice_io.get(MYCHOICE);
          text_io.skip_line;
         exit;
```

```
            exception
               when data_error | constraint_error =>
                   text_io.skip_line;
                   text_io.put_line("Your choice must be between 0 an ' 5");
                   text_io.new_line;
          end;
         end loop;
    --
    -- do different tasks from here
    --
      case MYCHOICE is
          when 0 =>
             HEAD := INITIAL_DB;
          when 1 =>
             DATA_RETRIEVE(HEAD);
          when 2 =>
             DATA_SCAN_IO(HEAD);
          when 3 =>
             DATA_SCAN(HEAD);
          when 4 =>
             exit;
          when others => null;
      end case;
    end loop;
end PUROBJ;
```

B.10 Test Program: purobj.c

```
/* C purobj.c program - main file
 A pure C program but perform the same functionality of
 adaobj.c except no persistent objects
*/
#include <stdio.h>
#include <strings.h>
#include <sys/time.h>
#include <sys/resource.h>

#include "purobj.h"


extern FILE *basic_file;
NOTE initial_db();
void data_retrieve();
void data_scan_io();
void data_scan();
NOTE database_retrieve();
void display_note();
void traverse();
void exit();
int  is_empty();

     /* global timing variables for CommandStats -- Jacobs 18/09/91 */
   static struct timeval elapsed;
   static struct rusage  exec;
main()
{
  char     choose;
  static NOTE head;

  head = NULL;
  /*-- main loop --
    drive manual
  -- choose input by user   */
 while (1) {
   printf("**    Testing Menu    **\n");
   printf(" 0. Initial database : transient only\n");
   printf(" 1. Testing the lookup and retrieve\n");
   printf(" 2. Testing the sequential scaning i/o\n");
   printf(" 3. Testing the sequential scaning without i/o\n");
   printf(" 4. Bye !!\n");
   printf("Input -> ");
  while (scanf("%c" , &choose) ==1) {
     if (choose <= '4' && choose >= '0')
          break;
      }
   if (choose == '4') exit(1);
```

```
    switch (choose) {
      case  '0' :
         head = initial_db();
         break;
      case  'i' :
         data_retrieve(head);
         break;
      case  '2' :
         data_scan_io(head);
         break;
      case  '3' :
         data_scan(head);
         break;

     }
   }
}

int is_empty(temp)
NOTE temp;
{
int empty;
 if (temp == NULL)
   empty = 1;
 else
   empty = 0;

 return empty;
 }

void data_retrieve(temp)
NOTE temp;
 {

int  input_number;
NOTE  basic_note;

printf("Retrive a record, the priority is  5000\n");
/*  if (scanf("%d", &input_number) == 1) */
   if (is_empty(temp))
      printf("\n%s\n","... LINK LIST IS EMPTY ...");
   else {
   input_number = 5000;
/* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     basic_note = database_retrieve(input_number, temp);
       display_note(basic_note);
/* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
      }
}
```

```
void data_scan_io(temp)
NOTE temp;
{
   if (is_empty(temp))
      printf("\n%s\n","... LINK LIST IS EMPTY ...");
   else
     {
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     traverse(temp,1);
    /* Stop time commandstat */
     CommandStats(0, stdout, &elapsed, &exec);
  }

}

void data_scan(temp)
NOTE temp;
{

   if (is_empty(temp))
      printf("\n%s\n","... LINK LIST IS EMPTY ...");
   else
     {
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
     traverse(temp,0);
    /* Stop time commandstat */
     CommandStats(0, stdout, &elapsed, &exec);
  }

}



/* retrieve a note  */
NOTE database_retrieve(numb, anote)
int numb;
NOTE anote;
{
NOTE temp;

 temp = anote;
 while ((numb != temp -> priority ) && (temp != NULL))
    temp = temp-> next;

 return temp;
 }
```

```
/* Print out to the specified stream this note */
void display_note(n)
NOTE n;
{
      printf("priority (%d)      name %s \n", n->priority, n->name);
      printf("note  %s\n", n -> note);
}

/* Insert a node   */
NOTE insert(p, q)
NOTE p;
NOTE q;
{
   q -> next= p;
   return q;
}



/* Sequencial scaning */
void traverse(anote,io)
NOTE anote;
int io;
{
NOTE temp;

 temp = anote;
 while (temp != NULL) {
   if (io != 0) {
       display_note(temp);
    }
  temp = temp -> next;
 }
}

NOTE initial_db()
{
     /* global timing variables for CommandStats -- Jacobs 18/09/91 */
    NOTE    head, basic_note;
    int     counter;

  head = NULL;
      /* Start time commandstat  */
  CommandStats(1, stdout, &elapsed, &exec);
    for (counter = 1; counter <= 10000; ++counter){
      basic_note = (NOTE) malloc(sizeof(struct link_note));
       switch (counter - ((int)(counter/10)) * 10) {
         case 0 :
          basic_note -> priority = counter ;
```

```
 strcpy(      note -> name ,"Danile");
 strcpy(basic_note -> note ,"you need meet your friend tomorrow");
 break;
case 1 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Susan");
 strcpy(basic_note -> note ,"you need meet Course commitee at 9:00");
 break;

case 2:
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Li");
 strcpy(basic_note -> note ,"Enjoy the silent night in Lab");
 break;

case 3 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Chars");
 strcpy(basic_note -> note ,"You may meet me at 11:00");
 break;
case 4 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Mike");
 strcpy(basic_note -> note ,"We have an appointment with principle");
 break;
case 5 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Coan");
 strcpy(basic_note -> note ,"We found a book you lost");
 break;
case 6 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Nancy");
 strcpy(basic_note -> note ,"Study Chapter 10 of OS");
 break;
case 7 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Patric");
 strcpy(basic_note -> note ,"Please collect class addresses.");
 break;
case 8 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Jenny");
 strcpy(basic_note -> note ,"Happy New Year");
 break;
case 9 :
 basic_note -> priority = counter ;
 strcpy(basic_note -> name ,"Amy");
 strcpy(basic_note -> note ,"Merry Christmas");
 break;
   }
```

```
    head = insert(head, basic_note);

    }
  /* Stop time commandstat */
    CommandStats(0, stdout, &elapsed, &exec);
  return head;
}
```

B.11 Test Program: hello_ost.mk (for hello_ost.a)

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

OS_COMPILATION_SCHEMA_DB_PATH = /lchou/hello.csdb
OS_APPLICATION_SCHEMA_DB_PATH = /lchou/hello.asdb
SCHEMA_SOURCE = hello_os.cc
LDLIBS   = -los -losc
CPPFLAGS = -gx -I..
EXCUTABLE = hello_ost
OBJECTS  = .os_schema.o hello_ost.o

hello_ost: .os_schema.o
a.make -L .. hello_ost -f hello_ost.a
mv a.out hello_ost
$(OS_ROOTDIR)/lib/patch hello_ost


include ../ada.mk
```

B.12 Test Program: hello_ost.a

```
with OS_TYPES; use OS_TYPES;
with OSTORE; use OSTORE;
with PERS_SCALARS;
with STATIS_ADA;
with TEXT_IO; use TEXT_IO;

procedure hello_ost is
pragma LINK_WITH("-Bstatic .os_schema.o ../libosada.a statis.o CommandStats.o
                  -L/usr/local/objectstore/sun4/lib -los -losc");
package INT_IO is new integer_io(INTEGER);
use INT_IO;
-- add check the performance
--
package time_io is new fixed_io(duration);
use Time_Io;
A_Number   : Integer;
Count      : Integer;
The_Choice : Character;

ROOT: DATABASE_ROOT;
IP: PERS_SCALARS.INTEGER_PTR;
DB: DATABASE;
TX: TRANSACTION;
begin
    put(" The numbers loop to perform -> ");
    get(A_number);
-- start time
   STATIS_ADA.COMMANDSTATS(1);
       INIT_ADA_INTERFACE;
       DB := DATABASE_OPEN("/lchou/ada.db", FALSE, 8#664#);
   for  Count in 1 .. A_number  loop

TX := TRANSACTION_BEGIN;
ROOT := DATABASE_ROOT_FIND("counter", DB);
if invalid(ROOT) then
ROOT := DATABASE_CREATE_ROOT(DB, "counter");
IP := PERS_SCALARS.PERS_INTEGER.PERSISTENT_NEW(DB);
PERS_SCALARS.PERS_INTEGER.DATABASE_ROOT_SET_VALUE(ROOT, IP);
end if;
IP := PERS_SCALARS.PERS_INTEGER.DATABASE_ROOT_GET_VALUE(ROOT);
IP.all := IP.all + 1;
put_line("Hello World!");
put("Program run now is               ");
put(IP.all);
put_line(" times.");
put("Program run from this exection is");
put(Count);
put_line(" times.");
TRANSACTION_COMMIT(TX);
   end loop;
```

```
-- stop time
   STATIS_ADA.COMMANDSTATS(0);

put("***  For performing ");
put(A_number,1);
put_line("  times ***");
new_line;
end HELLO_OST;
```

B.13 Test Program: hello_ost.mk (for hello_ost.c)

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk
OS_COMPILATION_SCHEMA_DB_PATH= /$(USER)/helloc.comp_schema
OS_APPLICATION_SCHEMA_DB_PATH= /$(USER)/helloc.app_schema
LDLIBS = -los -losc
SOURCES =   hello2at.c CommandStats.c schema.cc
OBJECTS =   hello2at.o hello2atb.o CommandStats.o schema.o
EXECUTABLES =   hello2at hello2atb
CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -g
CC = cc
LIB_PATH = -L/usr/local/objectstore/sun4/lib

all: $(EXECUTABLES)
###   using static binding ###

hello2at: hello2at.o CommandStats.o schema_standin_B
$(OS_PRELINK) .os_schema.cc \
    $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
    hello2at.o $(LDLIBS)
OSCC -c .os_schema.cc
$(LINK.c) -o hello2at -Bstatic hello2at.o CommandStats.o .os_schema.o \
        $(LDLIBS)
$(OS_POSTLINK) hello2at

hello2at.o: hello2at.c
$(CC) $(CPPFLAGS) $(CFLAGS) -c hello2at.c

CommandStats.o: CommandStats.c
cc $(CPPFLAGS) $(CFLAGS) -c CommandStats.c

schema_standin_B: schema.cc
OSCC $(CPPFLAGS) -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) schema.cc
touch schema_standin_B

clean:
osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
rm -f $(EXECUTABLES) $(OBJECTS) schema_standin_B

depend: .depend_B

.depend_B:
osmakedep .depend_B $(CPPFLAGS) -files $(SOURCES)

include .depend_B
```

B.14 Test Program: hello_ost.c

```
#include <stdio.h>
#include <ostore/ostore.h>
#include <sys/time.h>
#include <sys/resource.h>

  /* global timing variables for CommandStats -- Jacobs 18/09/91 */
 static struct timeval elapsed;
 static struct rusage  exec;

main( )
{
    database *db1;
    database_root *count_root;
    int *countp, counter, i;
    extern double get_clock( );
    double start_time, calculation_time;
    start_objectstore();

    printf(" The numbers loop to perform -> ");
    if (scanf("%d", &counter) == 1) {
    /* Start time commandstat  */
     CommandStats(1, stdout, &elapsed, &exec);
      db1 = database_lookup_open("/lchou/db1", 0, 0664);
      for (i = 1; i <= counter; ++i) {
         OS_BEGIN_TXN(tx1,0,transaction_update) {
    count_root = database_root_find("count", db1);
    countp = (int *)database_root_get_value(count_root, 0);

    printf("Hello, world.\n");
    printf("Program run.              %d times\n", ++*countp);
           printf("Run from this execution %d times\n", i);

      } OS_END_TXN(tx1);
     }
  /* Stop time commandstat */
   CommandStats(0, stdout, &elapsed, &exec);
     printf(" for performing %d times \n", counter);

   }
 }
```

Appendix C. Interface Programs

C.1 Interface Program: Makefile

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

SOURCES = glue.cc glue_pti.cc
OBJECTS = glue.o glue_pti.o

# set CC to your C++ compiler command
# unset TFLAGS (+OSTD is an OSCC flag to allow only standard C++)
TFLAGS=
CC=OSCC
CPPFLAGS = -gx
CFLAGS  = -g
CDEPEND = -I$(OS_ROOTDIR)/include

all: libosada.a ada_objects

clean:
osrm -f ${OS_COMPILATION_SCHEMA_DB_PATH}
rm -f ${EXECUTABLES} ${OBJECTS} schema_standin

glue.o: glue.cc
${CC} $(CPPFLAGS) $(TFLAGS) -c glue.cc

glue_pti.o: glue_pti.cc
$(CC) $(CPPFLAGS) $(TFLAGS) -c glue_pti.cc

libosada.a: glue.o glue_pti.o
ar rc libosada.a glue.o glue_pti.o
ranlib libosada.a

ada_objects:
a.make OS_TYPES           -f os_types.a os_typ_b.a
a.make OS_EXCEPTIONS      -f except.a   except_b.a
a.make OSTORE             -f ostore.a   ostore_b.a
a.make OSTORE_GENERICS    -f ostore_g.a ostorg_b.a
a.make OS_COLLECTION_PKG  -f os_coll.a  os_coll_b.a
a.make OS_CURSOR_PKG      -f os_cur.a   os_cur_b.a
a.make PERS_SCALARS       -f pscalr.a

.depend:
osmakedep .depend $(CDEPEND) -files $(SOURCES)

include .depend
```

C.2 Interface Program: os_types.a

```
--Definitions for objectstore's portable types

with UNSIGNED_TYPES;
package OS_TYPES is

subtype OS_UNSIGNED_INT8 is UNSIGNED_TYPES.UNSIGNED_TINY_INTEGER;
subtype OS_SIGNED_INT8 is TINY_INTEGER;
subtype OS_UNSIGNED_INT16 is UNSIGNED_TYPES.UNSIGNED_SHORT_INTEGER;
subtype OS_INT16 is SHORT_INTEGER;
subtype OS_UNSIGNED_INT32 is UNSIGNED_TYPES.UNSIGNED_INTEGER;
subtype OS_INT32 is INTEGER;
subtype OS_BOOLEAN is INTEGER;
subtype OS_UNIXTIME_T is UNSIGNED_TYPES.UNSIGNED_INTEGER;
subtype OS_COMPARE_RESULT is INTEGER;
subtype OS_BITF is UNSIGNED_TYPES.UNSIGNED_INTEGER;
type OSTORE_OPAQUE is private;
        type OS_STRING is new STRING(1..150);
        type OS_COLLECTION_BEHAVIOR is (MAINTAIN_CURSORS, ALLOW_DUPLICATES,
        SIGNAL_DUPLICATES, ALLOW_NULLS, MAINTAIN_ORDER);
        for OS_COLLECTION_BEHAVIOR'SIZE use 32;
function valid(OBJ: OSTORE_OPAQUE) return BOOLEAN;
function invalid(OBJ: OSTORE_OPAQUE) return BOOLEAN;

 private
type OSTORE_OPAQUE is new INTEGER;


end OS_TYPES;
```

C.3 Interface Program: os_typ_b.a

```
-- Implementation for os types
package body OS_TYPES is

function valid(OBJ: OSTORE_OPAQUE) return BOOLEAN is
begin
return OBJ /= 0;
end valid;
pragma INLINE(valid);

function invalid(OBJ: OSTORE_OPAQUE) return BOOLEAN is
begin
return OBJ = 0;
end invalid;
pragma INLINE(invalid);

end OS_TYPES;
```

C.4 Interface Program: ostore.a

```
-- Basic interface to ObjectStore from the Ada programming language
-- Prototype design and implementation by Dave Rosenberg of Object
-- Design, Inc.
--
--Functions are extended and  binding is changed to C library interface
--by Li Chou. Nov, 1992.
--

with SYSTEM; use SYSTEM;
with OS_TYPES; use OS_TYPES;
package OSTORE is

-- Public Types
  type STRPTR is access STRING;
  type DATABASE is new OSTORE_OPAQUE;
  type DATABASE_ROOT is new OSTORE_OPAQUE;
--collections
  type OS_COLLECTION is new OSTORE_OPAQUE;
  type OS_COLL_REP_DSCPR is new OSTORE_OPAQUE;
  type OS_CURSOR  is new OSTORE_OPAQUE;
  type SEGMENT is new OSTORE_OPAQUE;
  type OS_TYPESPEC is new OSTORE_OPAQUE;

-- transactions
  type TRANSACTION is new OSTORE_OPAQUE;
  type CONFIGURATION is new OSTORE_OPAQUE;
  type TRANSACTION_TYPE is (NONE, UPDATE, READ_ONLY);
  for TRANSACTION_TYPE'SIZE use 32;
  for TRANSACTION_TYPE use (NONE => 0, UPDATE => 1, READ_ONLY => 2);
  type REFERENCE is private;
  subtype U_MODE is OS_INT32 range 0 .. 8#777#;

-- Database Operations
  procedure DATABASE_CLOSE(DB: DATABASE);

  -- Raise ERR_DATABASE_EXISTS
  function DATABASE_CREATE(PATH: STRING; MODE: U_MODE := 8#664#;
   OVERWRITE: BOOLEAN := FALSE) return DATABASE;

  -- Raise ERR_DATABASE_NOT_FOUND
  function DATABASE_LOOKUP(PATH: STRING; MODE: U_MODE := 0) return DATABASE;

  -- Raise ERR_DATABASE_NOT_FOUND
  function DATABASE_OPEN(PATH: STRING; READ_ONLY: BOOLEAN := FALSE;
        MODE: U_MODE := 0) return DATABASE;

--  function DATABASE_GET_TRANSIENT_DATABASE return DATABASE;

--  function DATABASE_OF(LOC: ADDRESS) return DATABASE;
```

```
-- Objectstore Operations
    procedure INIT_ADA_INTERFACE;

    procedure START_OBJECTSTORE;


--  function OBJECTSTORE_IS_PERSISTENT(LOC: ADDRESS) return BOOLEAN;

--  procedure OBJECTSTORE_CHMOD(PATH: STRING; MODE: NATURAL);

--  procedure OBJECTSTORE_SET_BUFFER_SIZE(BYTES: POSITIVE);


-- Transaction Operations
  function TRANSACTION_GET_CURRENT return TRANSACTION;

  procedure TRANSACTION_ABORT(TX: TRANSACTION := TRANSACTION_GET_CURRENT);

  function TRANSACTION_GET_MAX_RETRIES return OS_INT32;

  procedure TRANSACTION_SET_MAX_RETRIES(COUNT: NATURAL := 10);

  function TRANSACTION_TOP_LEVEL(TX: TRANSACTION := TRANSACTION_GET_CURRENT)
   return BOOLEAN;

  function TRANSACTION_GET_TYPE(TX: TRANSACTION := TRANSACTION_GET_CURRENT)
    return TRANSACTION_TYPE;

  procedure TRANSACTION_COMMIT(TX: TRANSACTION := TRANSACTION_GET_CURRENT);

  procedure TRANSACTION_ABORT_TOP_LEVEL;

  function TRANSACTION_BEGIN(T_TYPE: TRANSACTION_TYPE := UPDATE)
   return TRANSACTION;

--  transaction_get_parent is not provided in C library
--  function TRANSACTION_GET_PARENT(TX: TRANSACTION := TRANSACTION_GET_CURRENT)
--    return TRANSACTION;

-- Utility functions
  function B_TO_OSB(V: BOOLEAN) return OS_BOOLEAN;


-- Database Root Operations
  -- Returns null if root not found!
  function DATABASE_ROOT_FIND(NAME: STRING; DB: DATABASE)
   return DATABASE_ROOT;

  -- Raise ERR_ROOT_EXISTS and ERR_DATABASE_NOT_FOUND
```

```
  function DATABASE_CREATE_ROOT(DB: DATABASE; NAME: STRING)
return DATABASE_ROOT;

  function DATABASE_ROOT_GET_NAME(ROOT: DATABASE_ROOT) return STRING;


function ALLOC_TYPESPEC(NAME: STRING) return OS_TYPESPEC ;

private
  type REFERENCE is
record
  SEGID: OS_INT32;
  OFFSET: OS_INT32;
  WORD0: OS_INT32;
  WORD1: OS_INT32;
  WORD2: OS_INT32;
end record;

end OSTORE;
```

C.5 Interface Program: ostore_b.a

```
--Body implementation for a prototype ObjectStore/Ada interface
--Design and implementation by Dave Rosenberg, Object Design, Inc,
--performed under contract to PRC, Dec-Jan, 1991-92.
--
--Functions are extended and  binding is changed to C library interface
--by Li Chou. Nov, 1992.
--
with SYSTEM; use SYSTEM;
with LANGUAGE; use LANGUAGE;
with OS_EXCEPTIONS;
with C_Strings;
with A_Strings;
package body OSTORE is

--Utility C string conversion
function c_ada_to_c(S: SYSTEM.ADDRESS; L:INTEGER) return SYSTEM.ADDRESS;
pragma INTERFACE(C, c_ada_to_c);
pragma INTERFACE_NAME(c_ada_to_c, C_SUBP_PREFIX & "c_ada_to_c");
-------------------------------------------------------------
--DATABASE_OPEN
function c_database_open(PATH: ADDRESS; OVERWRITE: OS_BOOLEAN;
 MODE: U_MODE) return DATABASE;
pragma INTERFACE(C, c_database_open);
pragma INTERFACE_NAME(c_database_open,
                      C_SUBP_PREFIX & "database_lookup_open");

function DATABASE_OPEN(PATH: STRING; READ_ONLY: BOOLEAN := FALSE;
       MODE: U_MODE := 0) return DATABASE is
begin
return c_database_open(c_ada_to_c(PATH(PATH'FIRST)'ADDRESS,
                           PATH'LENGTH), B_TO_OSB(READ_ONLY), MODE);
end DATABASE_OPEN;
pragma INLINE(DATABASE_OPEN);
-------------------------------------------------------------
--DATABASE_CLOSE
procedure c_database_close(DB : DATABASE);
pragma INTERFACE(C, c_database_close);
pragma INTERFACE_NAME(c_database_close,
                      C_SUBP_PREFIX & "database_close");

procedure DATABASE_CLOSE(DB : DATABASE) is
begin
     c_database_close(DB);
end DATABASE_CLOSE;
pragma INLINE(DATABASE_CLOSE);
-------------------------------------------------------------
--DATABASE_CREATE
function c_database_create(PATH: ADDRESS; MODE: U_MODE;
 OVERWRITE: OS_BOOLEAN) return DATABASE;
```

```
pragma INTERFACE(C, c_database_create);
pragma INTERFACE_NAME(c_database_create,
                      C_SUBP_PREFIX & "database_create");


function DATABASE_CREATE(PATH: STRING; MODE: U_MODE := 8#664#;
 OVERWRITE: BOOLEAN := FALSE) return DATABASE is
begin
return c_database_create(c_ada_to_c(PATH(PATH'FIRST)'ADDRESS,
                      PATH'LENGTH), MODE, B_TO_USB(OVERWRITE));
end DATABASE_CREATE;
pragma INLINE(DATABASE_CREATE);


--DATABASE_LOOKUP
function c_database_lookup(PATH: ADDRESS; MODE: U_MODE) return DATABASE;
pragma INTERFACE(C, c_database_lookup);
pragma INTERFACE_NAME(c_database_lookup,
                      C_SUBP_PREFIX & "database_lookup");

function DATABASE_LOOKUP(PATH: STRING; MODE: U_MODE := 0) return DATABASE is
begin

  return c_database_lookup(c_ada_to_c(PATH(PATH'FIRST)'ADDRESS,PATH'LENGTH),
                                                                  MODE);
end DATABASE_LOOKUP;
pragma INLINE(DATABASE_LOOKUP);
-------------------------------------------------------------
--Initialization
procedure c_init_ada_interface;
pragma INTERFACE(C, c_init_ada_interface);
pragma INTERFACE_NAME(c_init_ada_interface,
                      C_SUBP_PREFIX & "c_init_ada_interface");
procedure INIT_ADA_INTERFACE is
begin
c_init_ada_interface;
end INIT_ADA_INTERFACE;
pragma INLINE(INIT_ADA_INTERFACE);
-------------------------------------------------------------
--Initialization
procedure c_start_objectstore;
pragma INTERFACE(C, c_start_objectstore);
pragma INTERFACE_NAME(c_start_objectstore,
                      C_SUBP_PREFIX & "start_objectstore");
procedure START_OBJECTSTORE is
begin
c_start_objectstore;
end START_OBJECTSTORE;
pragma INLINE(START_OBJECTSTORE);
-------------------------------------------------------------------
--Transaction get current
```

```
function c_transaction_get_current return TRANSACTION;
pragma INTERFACE(C, c_transaction_get_current);
pragma INTERFACE_NAME(c_transaction_get_current,
                      C_SUBP_PREFIX & "transaction_get_current");
function TRANSACTION_GET_CURRENT return TRANSACTION is
begin
return c_transaction_get_current;
end TRANSACTION_GET_CURRENT;
pragma INLINE(TRANSACTION_GET_CURRENT);
-------------------------------------------------------------------
--Transaction begin
function c_transaction_begin(T: TRANSACTION_TYPE) return TRANSACTION;
pragma INTERFACE(C, c_transaction_begin);
pragma INTERFACE_NAME(c_transaction_begin,
           C_SUBP_PREFIX & "transaction_begin");
function TRANSACTION_BEGIN(T_TYPE: TRANSACTION_TYPE := UPDATE)
return TRANSACTION is
begin
return c_transaction_begin(T_TYPE);
end TRANSACTION_BEGIN;
pragma INLINE(TRANSACTION_BEGIN);
-------------------------------------------------------------
--Transaction commit
procedure c_transaction_commit(T: TRANSACTION);
pragma INTERFACE(C, c_transaction_commit);
pragma INTERFACE_NAME(c_transaction_commit,
                      C_SUBP_PREFIX & "transaction_commit");
procedure TRANSACTION_COMMIT(TX: TRANSACTION := TRANSACTION_GET_CURRENT) is
begin
c_transaction_commit(TX);
end TRANSACTION_COMMIT;
pragma INLINE(TRANSACTION_COMMIT);
-------------------------------------------------------------
--
procedure c_transaction_abort_top_level;
pragma INTERFACE(C, c_transaction_abort_top_level);
pragma INTERFACE_NAME(c_transaction_abort_top_level,
                      C_SUBP_PREFIX & "transaction_abort_top_level");
procedure TRANSACTION_ABORT_TOP_LEVEL is
begin
c_transaction_abort_top_level;
end TRANSACTION_ABORT_TOP_LEVEL;
pragma INLINE(TRANSACTION_ABORT_TOP_LEVEL);
-------------------------------------------------------------
--
procedure c_transaction_abort(T: TRANSACTION);
pragma INTERFACE(C, c_transaction_abort);
pragma INTERFACE_NAME(c_transaction_abort,
                      C_SUBP_PREFIX & "transaction_abort");
procedure TRANSACTION_ABORT(TX: TRANSACTION := TRANSACTION_GET_CURRENT) is
begin
```

```
c_transaction_abort(TX);
end TRANSACTION_ABORT;
pragma INLINE(TRANSACTION_ABORT);
-------------------------------------------------------------
--
function c_transaction_get_type(T: TRANSACTION) return TRANSACTION_TYPE;
pragma INTERFACE(C, c_transaction_get_type);
pragma INTERFACE_NAME(c_transaction_get_type,
                      C_SUBP_PREFIX & "transaction_get_type");
function TRANSACTION_GET_TYPE(TX: TRANSACTION := TRANSACTION_GET_CURRENT)
        return  TRANSACTION_TYPE is
begin
  return  c_transaction_get_type(TX);
end TRANSACTION_GET_TYPE;
pragma INLINE(TRANSACTION_GET_TYPE);
-------------------------------------------------------------
--
function c_transaction_get_max_retries  return OS_INT32;
pragma INTERFACE(C, c_transaction_get_max_retries);
pragma INTERFACE_NAME(c_transaction_get_max_retries,
                      C_SUBP_PREFIX & "transaction_get_max_retries");
function TRANSACTION_GET_MAX_RETRIES return  OS_INT32 is
begin
  return  c_transaction_get_max_retries;
end TRANSACTION_GET_MAX_RETRIES;
pragma INLINE(TRANSACTION_GET_MAX_RETRIES);
-------------------------------------------------------------
--
procedure c_transaction_set_max_retries(C  : NATURAL);
pragma INTERFACE(C, c_transaction_set_max_retries);
pragma INTERFACE_NAME(c_transaction_set_max_retries,
                      C_SUBP_PREFIX & "transaction_set_max_retries");
procedure TRANSACTION_SET_MAX_RETRIES(COUNT: NATURAL := 10) is
begin
c_transaction_set_max_retries(COUNT);
end TRANSACTION_SET_MAX_RETRIES;
pragma INLINE(TRANSACTION_SET_MAX_RETRIES);
-------------------------------------------------------------
--
function c_transaction_top_level(T: TRANSACTION) return BOOLEAN;
pragma INTERFACE(C, c_transaction_top_level);
pragma INTERFACE_NAME(c_transaction_top_level,
                      C_SUBP_PREFIX & "transaction_top_level");
function TRANSACTION_TOP_LEVEL(TX: TRANSACTION := TRANSACTION_GET_CURRENT)
        return  BOOLEAN is
begin
  return  c_transaction_top_level(TX);
end TRANSACTION_TOP_LEVEL;
pragma INLINE(TRANSACTION_TOP_LEVEL);
-------------------------------------------------------------
--Utility functions
```

```
-------------------------------------------------------------
function B_TO_OSB(V: BOOLEAN) return OS_BOOLEAN is
begin
if V then
return 1;
else
return 0;
end if;
end B_TO_OSB;
pragma INLINE(B_TO_OSB);

function OSB_TO_B(I: integer) return BOOLEAN is
begin
if I > 0 then
return true;
else
return false;
end if;
end OSB_TO_B;
pragma INLINE(OSB_TO_B);
-------------------------------------------------------------
--DATABASE ROOT FUNCTIONS
-------------------------------------------------------------
--DATABASE ROOT FIND

function c_database_root_find( A: ADDRESS; D: DATABASE)
return DATABASE_ROOT;
pragma INTERFACE(C, c_database_root_find);
pragma INTERFACE_NAME(c_database_root_find,
                      C_SUBP_PREFIX & "database_root_find");
function DATABASE_ROOT_FIND(NAME: STRING; DB: DATABASE)
return DATABASE_ROOT is
begin
   return c_database_root_find(c_ada_to_c(NAME(NAME'FIRST)'ADDRESS,
                               NAME'LENGTH), DB);
end DATABASE_ROOT_FIND;
pragma INLINE(DATABASE_ROOT_FIND);
-------------------------------------------------------------
--DATABASE CREATE ROOT
function c_database_create_root(D: DATABASE; A: ADDRESS)
return DATABASE_ROOT;
pragma INTERFACE(C, c_database_create_root);
pragma INTERFACE_NAME(c_database_create_root,
                      C_SUBP_PREFIX & "database_create_root");
function DATABASE_CREATE_ROOT(DB: DATABASE; NAME: STRING)
return DATABASE_ROOT is
begin
return c_database_create_root(DB,
           c_ada_to_c(NAME(NAME'FIRST)'ADDRESS, NAME'LENGTH));

end DATABASE_CREATE_ROOT;
```

```
pragma INLINE(DATABASE_CREATE_ROOT);
-------------------------------------------------------------
--DATABASE_ROOT_GET_NAME

function c_database_root_get_name(R : DATABASE_ROOT) return SYSTEM.ADDRESS;
pragma INTERFACE(C, c_database_root_get_name);
pragma INTERFACE_NAME(c_database_root_get_name,
                      C_SUBP_PREFIX & "database_root_get_name");


function DATABASE_ROOT_GET_NAME(ROOT : DATABASE_ROOT) return STRING is
ROOT_ADDRESS : SYSTEM.ADDRESS;
ROOT_NAME    : A_STRINGS.A_STRING;
NAME         : STRING(1..254) :=(others => ' ');
LEN          : natural := 0;

begin
  ROOT_ADDRESS :=c_database_root_get_name(ROOT);

  ROOT_NAME := C_Strings.CONVERT_C_TO_A(C_strings.to_c(ROOT_ADDRESS)) ;
  LEN    := ROOT_NAME.S'LENGTH;
  NAME(1.. LEN) := ROOT_NAME.S(1..LEN);
  return NAME(1.. LEN);
end DATABASE_ROOT_GET_NAME;
-------------------------------------------------------------
-- create new os_types
function C_ALLOC_TYPESPEC(A: address; I: integer) return OS_TYPESPEC;
   pragma INTERFACE(C, c_alloc_typespec);
   pragma INTERFACE_NAME(c_alloc_typespec ,
                      C_SUBP_PREFIX & "alloc_typespec");

function ALLOC_TYPESPEC(NAME: STRING) return OS_TYPESPEC is
begin
  return C_ALLOC_TYPESPEC(c_ada_to_c(NAME(NAME'FIRST)'ADDRESS, NAME'LENGTH),0);
end ALLOC_TYPESPEC;
pragma INLINE(ALLOC_TYPESPEC);

end OSTORE;
```

C.6 Interface Program: ostore_g.a

```
-- Basic interface to ObjectStore from the Ada programming language
-- Prototype design and implementation by Dave Rosenberg of Object
-- Design, Inc.  This file provides suitable generic definitons.
--
--Functions are extended and  binding is changed to C library interface
--by Li Chou. Nov, 1992.
--

with OS_TYPES; use OS_TYPES;
with OSTORE; use OSTORE;
with SYSTEM; use SYSTEM;
generic
  type U_TYPE is private;
  type U_TYPEPTR is access U_TYPE;
  with function GET_OS_TYPESPEC return OS_TYPESPEC;
package OSTORE_GENERICS is

-- Database Roots
  function DATABASE_ROOT_GET_VALUE(ROOT: DATABASE_ROOT) return U_TYPEPTR;

-- for collections -by Li Chou 92-11-06
  function DATABASE_ROOT_GET_VALUE(ROOT: DATABASE_ROOT) return OS_COLLECTION;

  procedure DATABASE_ROOT_SET_VALUE(ROOT: DATABASE_ROOT; VALUE: U_TYPEPTR);

--for collecitons  -by Li Chou 92-11-06
  procedure DATABASE_ROOT_SET_VALUE(ROOT: DATABASE_ROOT; VALUE:
  OS_COLLECTION);

-- Persistent Allocation
  function PERSISTENT_NEW(DB: DATABASE) return U_TYPEPTR;

end OSTORE_GENERICS;
```

C.7 Interface Program: ostorg_b.a

```
--Ada implementation for generic components of the ObjectStore interface

with OSTORE; use OSTORE;
with SYSTEM; use SYSTEM;
with LANGUAGE; use LANGUAGE;
with OS_TYPES; use OS_TYPES;
with OS_EXCEPTIONS;

package body OSTORE_GENERICS is

EXCEPTION_INX : OS_EXCEPTIONS.OS_EXCEPTION_INDEX;

--DATABASE ROOT GET VALUE
-- for os_collection. Li Chou 92-11-06
function c_database_root_get_value(R: DATABASE_ROOT; T: OS_TYPESPEC)
return OS_COLLECTION;

function c_database_root_get_value(R: DATABASE_ROOT; T: OS_TYPESPEC)
return U_TYPEPTR;
pragma INTERFACE(C, c_database_root_get_value);
pragma INTERFACE_NAME(c_database_root_get_value,
                      C_SUBP_PREFIX & "database_root_get_value");
function DATABASE_ROOT_GET_VALUE(ROOT: DATABASE_ROOT)
return U_TYPEPTR is
begin
   return c_database_root_get_value(ROOT,GET_OS_TYPESPEC);
end DATABASE_ROOT_GET_VALUE;
pragma INLINE(DATABASE_ROOT_GET_VALUE);

-- for collections -by Li Chou 92-11-06
function DATABASE_ROOT_GET_VALUE(ROOT: DATABASE_ROOT) return OS_COLLECTION is
begin
   return c_database_root_get_value(ROOT,GET_OS_TYPESPEC);
end DATABASE_ROOT_GET_VALUE;

--DATABASE ROOT SET VALUE
--for collecitons  -by Li Chou 92-11-06
procedure c_database_root_set_value(R: DATABASE_ROOT; V: OS_COLLECTION;
    T: OS_TYPESPEC);

procedure c_database_root_set_value(R: DATABASE_ROOT; V: U_TYPEPTR;
    T: OS_TYPESPEC);
pragma INTERFACE(C, c_database_root_set_value);
pragma INTERFACE_NAME(c_database_root_set_value,
                      C_SUBP_PREFIX & "database_root_set_value");
procedure DATABASE_ROOT_SET_VALUE(ROOT: DATABASE_ROOT; VALUE: U_TYPEPTR) is
begin
c_database_root_set_value(ROOT, VALUE, GET_OS_TYPESPEC);
end DATABASE_ROOT_SET_VALUE;
pragma INLINE(DATABASE_ROOT_SET_VALUE);
```

```
--for collecitons  -by Li Chou 92-11-06
  procedure DATABASE_ROOT_SET_VALUE(ROOT: DATABASE_ROOT; VALUE: OS_COLLECTION) is
begin
c_database_root_set_value(ROOT, VALUE, GET_OS_TYPESPEC);
end DATABASE_ROOT_SET_VALUE;

-- Persistent new
function c_persistent_new(T: OS_TYPESPEC; N: OS_INT32; DB: DATABASE)
return U_TYPEPTR;
pragma INTERFACE(C, c_persistent_new);
pragma INTERFACE_NAME(c_persistent_new,
     C_SUBP_PREFIX & "objectstore_alloc");
function PERSISTENT_NEW(DB: DATABASE) return U_TYPEPTR is
begin
return c_persistent_new(GET_OS_TYPESPEC, 1, DB);
end PERSISTENT_NEW;
pragma INLINE(PERSISTENT_NEW);

end OSTORE_GENERICS;
```

C.8 Interface Program: os_coll.a

```
--
-- Basic collection interface to ObjectStore from the Ada programming
-- language prototype  implementation by Li Chou
--
with SYSTEM; use SYSTEM;
with OSTORE; use OSTORE;
with OS_TYPES; use OS_TYPES;
generic
  type U_TYPE is private;
  type U_TYPEPTR is access U_TYPE;
  with function GET_OS_TYPESPEC return OS_TYPESPEC;

package OS_COLLECTION_PKG is

FUNCTION OS_COLLECTION_CHANGE_BEHAVIOR(OS_COL    : OS_COLLECTION;
                                       BEHV      : STRING;
                                       VERIFY    : BOOLEAN :=TRUE)
                                    RETURN OS_COLLECTION;

--  Collection Operations
--  parameters of os_coll_rep_descriptor and int (retain policy descriptor)
--  are not allowed in this function
--
-- create collection with behavior
-- 92-11-16
function OS_COLLECTION_CREATE(DB        : DATABASE;
                              BEHV      : STRING;
                              SIZE      : OS_INT32 := 0;
                            RETAIN      : BOOLEAN:= false
                               ) return OS_COLLECTION;

function OS_COLLECTION_CREATE(DB        : DATABASE;
                              SIZE      : OS_INT32 := 0
                               ) return OS_COLLECTION;


procedure OS_COLLECTION_DELETE(OS_COL : OS_COLLECTION);

function OS_COLLECTION_CARDINALITY(OS_COL : OS_COLLECTION) return
                                       OS_UNSIGNED_INT32;

procedure OS_COLLECTION_CLEAR(OS_COL : OS_COLLECTION);

function OS_COLLECTION_CONTAINS(OS_COL : OS_COLLECTION;
                                VALUE  : U_TYPEPTR)  return BOOLEAN;

procedure OS_COLLECTION_COPY(OS_COL_A : OS_COLLECTION;
                             OS_COL_B : OS_COLLECTION);

function OS_COLLECTION_COUNT(OS_COL    :  OS_COLLECTION;
```

```
                                VALUE    :  U_TYPEPTR) return OS_UNSIGNED_INT32;

procedure OS_COLLECTION_DIFFERENCE(OS_COL_A : OS_COLLECTION;
                                   OS_COL_B : OS_COLLECTION);

function OS_COLLECTION_EMPTY(OS_COL : OS_COLLECTION)  return BOOLEAN;

function OS_COLLECTION_EQUAL(OS_COL_A : OS_COLLECTION;
                             OS_COL_B : OS_COLLECTION) return BOOLEAN;

function OS_COLLECTION_GET_BEHAVIOR(OS_COL   :  OS_COLLECTION)
                                   return OS_UNSIGNED_INT32;

function OS_COLLECTION_GREATER_THAN(OS_COL_A : OS_COLLECTION;
                                   OS_COL_B : OS_COLLECTION) return BOOLEAN;

function OS_COLLECTION_GREATER_THAN_OR_EQUAL(
                             OS_COL_A : OS_COLLECTION;
                             OS_COL_B : OS_COLLECTION) return BOOLEAN;

-- os_collections
procedure OS_COLLECTION_INITIALIZE;

-- the functions of insert
procedure OS_COLLECTION_INSERT(OS_COL   :  OS_COLLECTION;
                               VALUE    :  U_TYPEPTR) ;

procedure OS_COLLECTION_INSERT_FIRST(OS_COL   :  OS_COLLECTION;
                                    VALUE    :  U_TYPEPTR) ;

procedure OS_COLLECTION_INSERT_LAST(OS_COL   :  OS_COLLECTION;
                                   VALUE    :  U_TYPEPTR) ;

procedure OS_COLLECTION_INTERSECT(OS_COL_A : OS_COLLECTION;
                                  OS_COL_B : OS_COLLECTION);

function OS_COLLECTION_LESS_THAN(OS_COL_A : OS_COLLECTION;
                                 OS_COL_B : OS_COLLECTION) return BOOLEAN;

function OS_COLLECTION_LESS_THAN_OR_EQUAL(OS_COL_A : OS_COLLECTION;
                                          OS_COL_B : OS_COLLECTION)
                                    return BOOLEAN;

function OS_COLLECTION_NOT_EQUAL(OS_COL_A : OS_COLLECTION;
                                 OS_COL_B : OS_COLLECTION) return BOOLEAN;

-- the functions of remove
function OS_COLLECTION_REMOVE(OS_COL  :  OS_COLLECTION;
                              VALUE   :  U_TYPEPTR) return BOOLEAN;

function OS_COLLECTION_REMOVE_FIRST(OS_COL : OS_COLLECTION) return U_TYPEPTR;
```

```
function OS_COLLECTION_REMOVE_LAST(OS_COL : OS_COLLECTION)  return U_TYPEPTR;

function OS_COLLECTION_ONLY(OS_COL : OS_COLLECTION)  return U_TYPEPTR;

function OS_COLLECTION_ORDERED_EQUAL(OS_COL_A : OS_COLLECTION,
                                     OS_COL_B : OS_COLLECTION)
                                   return BOOLEAN;

-- the functions of pick
function OS_COLLECTION_PICK(OS_COL : OS_COLLECTION) return U_TYPEPTR ;

function OS_COLLECTION_QUERY(OS_COL         : OS_COLLECTION;
                             ELEMENT_TYPE   : STRING;
                             EXPRESS_STRING : STRING;
                             DB             : DATABASE) RETURN OS_COLLECTION;

function OS_COLLECTION_QUERY_EXISTS(OS_COL  : OS_COLLECTION;
                             ELEMENT_TYPE   : STRING;
                             EXPRESS_STRING : STRING;
                             DB             : DATABASE) RETURN BOOLEAN;


function OS_COLLECTION_QUERY_PICK(OS_COL    : OS_COLLECTION;
                             ELEMENT_TYPE   : STRING;
                             EXPRESS_STRING : STRING;
                             DB             : DATABASE) RETURN U_TYPEPTR;

procedure OS_COLLECTION_UNION(OS_COL_A : OS_COLLECTION;
                              OS_COL_B : OS_COLLECTION);

end OS_COLLECTION_PKG;
```

C.9 Interface Program: os_coll_b.a

```
-- Basic collection interface to ObjectStore from the Ada programming
-- language prototype  implementation by Li Chou
--
with UNSIGNED_TYPES; use UNSIGNED_TYPES;
with LANGUAGE; use LANGUAGE;
with OSTORE; use OSTORE;
with OS_TYPES; use OS_TYPES;
with SYSTEM; use SYSTEM;
-- library provide by VERDIX ADA.
with A_STRINGS;

package body OS_COLLECTION PKG is

  SUBTYPE Uppercase_Character IS CHARACTER RANGE 'A' .. 'Z';
  SUBTYPE Lowercase_Character IS CHARACTER RANGE 'a' .. 'z';

 FUNCTION Is_Alphabetic (The_Character : IN Character) RETURN BOOLEAN IS

  BEGIN
    IF The_Character IN Uppercase_Character THEN
      RETURN TRUE;
    ELSIF The_Character IN Lowercase_Character THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;
    END IF;
  END Is_Alphabetic;

--  Collection Operations
--  parameters of os_coll_rep_descriptor and int (retain policy descriptor)
--  are not allowed in this function
--
-- create collection with behavior
-- 92-11-16
function ADA_OS_BEHAVIOR(BEHV : STRING)  return OS_UNSIGNED_INT32 is
position_counter : natural := 1;
behavior_counter : natural;
os_string_length : natural := 0;
os_behavior      : os_unsigned_int32 := 0;
temp_string    : string(1..30);
temp_behavior  : os_collection_behavior;
begin
    os_string_length := behv'length;
    while position_counter <= os_string_length loop
       if Is_Alphabetic(behv(position_counter)) then
         temp_string(1..30) :=(others => ' ');
         behavior_counter := 1;
         while  POSITION_COUNTER <= OS_STRING_LENGTH and then
            BEHV(POSITION_COUNTER) /= ' '
          loop
```

```
            TEMP_STRING(BEHAVIOR_COUNTER) := BEHV(POSITION_COUNTER);
            POSITION_COUNTER := POSITION_COUNTER + 1;
            BEHAVIOR_COUNTER := BEHAVIOR_COUNTER + 1;
          end loop; -- get behavior
          TEMP_BEHAVIOR := OS_COLLECTION_BEHAVIOR'VALUE(TEMP_STRING
           (1..temp_string'length));
          if temp_behavior in os_collection_behavior then
              os_behavior := os_behavior + 2 **
                    os_collection_behavior'pos(temp_behavior);
          end if;

          -- get ride of symbol '|' and space
        else
           position_counter := position_counter + 1 ;
        end if;
      end locp;
      return os_behavior;
end ADA_OS_BEHAVIOR;
--
FUNCTION C_OS_COLLECTION_CHANGE_BEHAVIOR(OS_COL   : OS_COLLECTION;
                              BEHAVIOR : OS_UNSIGNED_INT32;
                              VERIFY   : OS_BOOLEAN) RETURN OS_COLLECTION;

  PRAGMA INTERFACE(C, c_os_collection_change_behavior);
  PRAGMA INTERFACE_NAME(c_os_collection_change_behavior,
                        C_SUBP_PREFIX & "os_collection_change_behavior");

FUNCTION OS_COLLECTION_CHANGE_BEHAVIOR(OS_COL   : OS_COLLECTION;
                                       BEHV     : STRING;
                                       VERIFY   : BOOLEAN :=TRUE) RETURN
                                  OS_COLLECTION IS
COLL_BHV : OS_UNSIGNED_INT32 :=0;
BEGIN
  if BEHV'length /= 0 then
     COLL_BHV := ADA_OS_BEHAVIOR(BEHV(1..BEHV'length));
  end if;
  RETURN C_OS_COLLECTION_CHANGE_BEHAVIOR(OS_COL,COLL_BHV,B_TO_OSB(VERIFY));
END OS_COLLECTION_CHANGE_BEHAVIOR;
PRAGMA INLINE(OS_COLLECTION_CHANGE_BEHAVIOR);
--
function C_OS_COLLECTION_CREATE(DB       : DATABASE;
                              BEHAVIOR : OS_UNSIGNED_INT32;
                              SIZE     : OS_INT32;
                              DESCRIPTOR :OS_INT32;
                              RETAIN   : OS_BOOLEAN) return OS_COLLECTION;

  pragma INTERFACE(C, c_os_collection_create);
  pragma INTERFACE_NAME(c_os_collection_create,
                        C_SUBP_PREFIX & "os_collection_create");

function OS_COLLECTION_CREATE(DB       : DATABASE;
```

```
                              SIZE     : OS_INT32 := 0) return OS_COLLECTION is
-- behavor bitwise operation
-- os_collection_allow_nulls       => 1
-- os_collection_allow_duplicates  => 2
-- os_collection_signal_duplicates => 4
-- os_collection_maintain_order    => 8
-- os_collection_maintain_cursors  => 16
-- this sample default that the behavor is maintain_order and cursors (24)

COLL_BHV    : OS_UNSIGNED_INT32 := 24;
RETAIN      : BOOLEAN:= false;
begin

  return C_OS_COLLECTION_CREATE(DB,COLL_BHV,SIZE,0,B_TO_OSB(RETAIN));
end OS_COLLECTION_CREATE;
pragma INLINE(OS_COLLECTION_CREATE);
--
--
-- create collection with behavior
-- 92-11-16
function OS_COLLECTION_CREATE(DB       : DATABASE;
                              BEHV     : STRING;
                              SIZE     : OS_INT32 := 0;
                            RETAIN     : BOOLEAN:= false) return
                            OS_COLLECTION   is
COLL_BHV    : OS_UNSIGNED_INT32 := 0;
begin
  if BEHV'length /= 0 then
     COLL_BHV := ADA_OS_BEHAVIOR(BEHV(1..BEHV'length));
  end if;
  return C_OS_COLLECTION_CREATE(DB,COLL_BHV,SIZE,0,B_TO_OSB(RETAIN));
end OS_COLLECTION_CREATE;

procedure C_OS_COLLECTION_DELETE(OS_COL : OS_COLLECTION);
  pragma INTERFACE(C, c_os_collection_delete);
  pragma INTERFACE_NAME(c_os_collection_delete,
                        C_SUBP_PREFIX & "os_collection_delete");

procedure OS_COLLECTION_DELETE(OS_COL : OS_COLLECTION) is
begin
    C_OS_COLLECTION_DELETE(OS_COL);
end OS_COLLECTION_DELETE;
--
function c_os_collection_cardinality(OS_COL    :  OS_COLLECTION) return
                                     OS_UNSIGNED_INT32;
  pragma INTERFACE(C, c_os_collection_cardinality);
  pragma INTERFACE_NAME(c_os_collection_cardinality,
                        C_SUBP_PREFIX & "os_collection_cardinality");

function OS_COLLECTION_CARDINALITY(OS_COL : OS_COLLECTION ) return
                                   OS_UNSIGNED_INT32 is
```

```
begin
   return  c_os_collection_cardinality(OS_COL);
end OS_COLLECTION_CARDINALITY;
pragma INLINE(OS_COLLECTION_CARDINALITY);
--
procedure C_OS_COLLECTION_CLEAR(OS_COL : OS_COLLECTION);
  pragma INTERFACE(C, c_os_collection_clear);
  pragma INTERFACE_NAME(c_os_collection_clear,
                        C_SUBP_PREFIX & "os_collection_clear");

procedure OS_COLLECTION_CLEAR(OS_COL : OS_COLLECTION) is
begin
    C_OS_COLLECTION_CLEAR(OS_COL);
end OS_COLLECTION_CLEAR;
--
-- os_collection_contains
function C_OS_COLLECTION_CONTAINS(OS_COL : OS_COLLECTION;
                                  VALUE  : U_TYPEPTR) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_contains);
  pragma INTERFACE_NAME(c_os_collection_contains,
                        C_SUBP_PREFIX & "os_collection_contains");

function OS_COLLECTION_CONTAINS(OS_COL : OS_COLLECTION;
                                VALUE  : U_TYPEPTR )  return BOOLEAN is
begin
  return C_OS_COLLECTION_CONTAINS(OS_COL, VALUE);
end OS_COLLECTION_CONTAINS;
pragma INLINE(OS_COLLECTION_CONTAINS);
--
procedure C_OS_COLLECTION_COPY(OS_COL_A : OS_COLLECTION;
                               OS_COL_B : OS_COLLECTION);

  pragma INTERFACE(C, c_os_collection_copy);
  pragma INTERFACE_NAME(c_os_collection_copy,
                        C_SUBP_PREFIX & "os_collection_copy");

procedure OS_COLLECTION_COPY(OS_COL_A : OS_COLLECTION;
                             OS_COL_B : OS_COLLECTION)  is
begin
   C_OS_COLLECTION_COPY(OS_COL_A,OS_COL_B);
end OS_COLLECTION_COPY;
pragma INLINE(OS_COLLECTION_COPY);
--
function C_OS_COLLECTION_COUNT(OS_COL   :  OS_COLLECTION;
                               VALUE    :  U_TYPEPTR) return OS_UNSIGNED_INT32;
  pragma INTERFACE(C, c_os_collection_count);
  pragma INTERFACE_NAME(c_os_collection_count,
                        C_SUBP_PREFIX & "os_collection_count");
function OS_COLLECTION_COUNT(OS_COL   :  OS_COLLECTION;
                             VALUE    :  U_TYPEPTR) return OS_UNSIGNED_INT32 is
```

```
begin
   return  c_os_collection_count(OS_COL,VALUE);
end OS_COLLECTION_COUNT;
pragma INLINE(OS_COLLECTION_COUNT);
--
procedure C_OS_COLLECTION_DIFFERENCE(OS_COL_A : OS_COLLECTION;
                                     OS_COL_B : OS_COLLECTION);

  pragma INTERFACE(C, c_os_collection_difference);
  pragma INTERFACE_NAME(c_os_collection_difference,
                        C_SUBP_PREFIX & "os_collection_difference");


procedure OS_COLLECTION_DIFFERENCE(OS_COL_A : OS_COLLECTION;
                                   OS_COL_B : OS_COLLECTION)  is
begin
   C_OS_COLLECTION_DIFFERENCE(OS_COL_A,OS_COL_B);
end OS_COLLECTION_DIFFERENCE;
pragma INLINE(OS_COLLECTION_DIFFERENCE);
--
--
-- os_collection_empty
function C_OS_COLLECTION_EMPTY(OS_COL : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_empty);
  pragma INTERFACE_NAME(c_os_collection_empty,
                        C_SUBP_PREFIX & "os_collection_empty");


function OS_COLLECTION_EMPTY(OS_COL : OS_COLLECTION)  return BOOLEAN is
begin
  return C_OS_COLLECTION_EMPTY(OS_COL);
end OS_COLLECTION_EMPTY;
pragma INLINE(OS_COLLECTION_EMPTY);
--
--
-- os_collection_equal
function C_OS_COLLECTION_EQUAL(OS_COL_A : OS_COLLECTION;
                               OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_equal);
  pragma INTERFACE_NAME(c_os_collection_equal,
                        C_SUBP_PREFIX & "os_collection_equal");


function OS_COLLECTION_EQUAL(OS_COL_A : OS_COLLECTION;
                             OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_EQUAL(OS_COL_A,OS_COL_B);
end OS_COLLECTION_EQUAL;
pragma INLINE(OS_COLLECTION_EQUAL);
```

```
--
-- os_collection_get_behavior
function c_os_collection_get_behavior(OS_COL    :  OS_COLLECTION
                                     ) return OS_UNSIGNED_INT32;
  pragma INTERFACE(C, c_os_collection_get_behavior);
  pragma INTERFACE_NAME(c_os_collection_get_behavior,
                        C_SUBP_PREFIX & "os_collection_get_behavior");
function OS_COLLECTION_GET_BEHAVIOR(OS_COL    :  OS_COLLECTION
                                   ) return OS_UNSIGNED_INT32 is
begin
   return  c_os_collection_get_behavior(OS_COL);
end OS_COLLECTION_GET_BEHAVIOR;
pragma INLINE(OS_COLLECTION_GET_BEHAVIOR);
--
-- os_collection_greater_than
function C_OS_COLLECTION_GREATER_THAN(OS_COL_A : OS_COLLECTION;
                                      OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_greater_than);
  pragma INTERFACE_NAME(c_os_collection_greater_than,
                        C_SUBP_PREFIX & "os_collection_greater_than");


function OS_COLLECTION_GREATER_THAN(OS_COL_A : OS_COLLECTION;
                                    OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_GREATER_THAN(OS_COL_A,OS_COL_B);
end OS_COLLECTION_GREATER_THAN;
pragma INLINE(OS_COLLECTION_GREATER_THAN);
--
-- os_collection_greater_than_or_equal
function C_OS_COLLECTION_GREATER_THAN_OR_EQUAL(OS_COL_A : OS_COLLECTION;
                                      OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_greater_than_or_equal);
  pragma INTERFACE_NAME(c_os_collection_greater_than_or_equal,
                        C_SUBP_PREFIX & "os_collection_greater_than_or_equal");


function OS_COLLECTION_GREATER_THAN_OR_EQUAL(OS_COL_A : OS_COLLECTION;
                                    OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_GREATER_THAN_OR_EQUAL(OS_COL_A,OS_COL_B);
end OS_COLLECTION_GREATER_THAN_OR_EQUAL;
pragma INLINE(OS_COLLECTION_GREATER_THAN_OR_EQUAL);

-- implemented by Li Chou. 92-11-04
-- os_collections
procedure C_OS_COLLECTION_INITIALIZE;
  pragma INTERFACE(C, c_os_collection_initialize);
  pragma INTERFACE_NAME(c_os_collection_initialize,
```

```
                        C_SUBP_PREFIX & "os_collection_initialize");
procedure OS_COLLECTION_INITIALIZE is
begin
   C_OS_COLLECTION_INITIALIZE;
end OS_COLLECTION_INITIALIZE;
pragma INLINE(OS_COLLECTION_INITIALIZE);


--
-- the functions of insert
--
procedure C_OS_COLLECTION_INSERT(OS_COL  :  OS_COLLECTION;
                                 VALUE   :  U_TYPEPTR);
  pragma INTERFACE(C, c_os_collection_insert);
  pragma INTERFACE_NAME(c_os_collection_insert,
                        C_SUBP_PREFIX & "os_collection_insert");

procedure OS_COLLECTION_INSERT(OS_COL  :  OS_COLLECTION;
                               VALUE   :  U_TYPEPTR) is
begin
  C_OS_COLLECTION_INSERT(OS_COL,VALUE);
end OS_COLLECTION_INSERT;
pragma INLINE(OS_COLLECTION_INSERT);
--
-- OS_COLLECTION_INSERT_FIRST
procedure C_OS_COLLECTION_INSERT_FIRST(OS_COL  :  OS_COLLECTION;
                                       VALUE   :  U_TYPEPTR);
  pragma INTERFACE(C, c_os_collection_insert_first);
  pragma INTERFACE_NAME(c_os_collection_insert_first,
                        C_SUBP_PREFIX & "os_collection_insert_first");

procedure OS_COLLECTION_INSERT_FIRST(OS_COL :  OS_COLLECTION;
                                     VALUE  :  U_TYPEPTR) is
begin
  C_OS_COLLECTION_INSERT_FIRST(OS_COL,VALUE);
end OS_COLLECTION_INSERT_FIRST;
pragma INLINE(OS_COLLECTION_INSERT_FIRST);


--
-- os_collection_insert_last
procedure C_OS_COLLECTION_INSERT_LAST(OS_COL  :  OS_COLLECTION;
                                      VALUE   :  U_TYPEPTR);
  pragma INTERFACE(C, c_os_collection_insert_last);
  pragma INTERFACE_NAME(c_os_collection_insert_last,
                        C_SUBP_PREFIX & "os_collection_insert_last");

procedure OS_COLLECTION_INSERT_LAST(OS_COL :  OS_COLLECTION;
                                    VALUE  :  U_TYPEPTR) is
begin
  C_OS_COLLECTION_INSERT_LAST(OS_COL,VALUE);
```

```
end OS_COLLECTION_INSERT_LAST;
pragma INLINE(OS_COLLECTION_INSERT_LAST);
--
--
procedure C_OS_COLLECTION_INTERSECT(OS_COL_A : OS_COLLECTION;
                                    OS_COL_B : OS_COLLECTION);

  pragma INTERFACE(C, c_os_collection_intersect);
  pragma INTERFACE_NAME(c_os_collection_intersect,
                        C_SUBP_PREFIX & "os_collection_intersect");


procedure OS_COLLECTION_INTERSECT(OS_COL_A : OS_COLLECTION;
                                  OS_COL_B : OS_COLLECTION)  is
begin
    C_OS_COLLECTION_INTERSECT(OS_COL_A,OS_COL_B);
end OS_COLLECTION_INTERSECT;
pragma INLINE(OS_COLLECTION_INTERSECT);
--
-- os_collection_less_than
function C_OS_COLLECTION_LESS_THAN(OS_COL_A : OS_COLLECTION;
                                   OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_less_than);
  pragma INTERFACE_NAME(c_os_collection_less_than,
                        C_SUBP_PREFIX & "os_collection_less_than");


function OS_COLLECTION_LESS_THAN(OS_COL_A : OS_COLLECTION;
                                 OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_LESS_THAN(OS_COL_A,OS_COL_B);
end OS_COLLECTION_LESS_THAN;
pragma INLINE(OS_COLLECTION_LESS_THAN);
--
-- os_collection_less_than_or_equal
function C_OS_COLLECTION_LESS_THAN_OR_EQUAL(OS_COL_A : OS_COLLECTION;
                                   OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_less_than_or_equal);
  pragma INTERFACE_NAME(c_os_collection_less_than_or_equal,
                        C_SUBP_PREFIX & "os_collection_less_than_or_equal");


function OS_COLLECTION_LESS_THAN_OR_EQUAL(OS_COL_A : OS_COLLECTION;
                                 OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_LESS_THAN_OR_EQUAL(OS_COL_A,OS_COL_B);
end OS_COLLECTION_LESS_THAN_OR_EQUAL;
pragma INLINE(OS_COLLECTION_LESS_THAN_OR_EQUAL);
```

```
--
-- os_collection_not_equal
function C_OS_COLLECTION_NOT_EQUAL(OS_COL_A : OS_COLLECTION;
                                   OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_not_equal);
  pragma INTERFACE_NAME(c_os_collection_not_equal,
                        C_SUBP_PREFIX & "os_collection_not_equal");


function OS_COLLECTION_NOT_EQUAL(OS_COL_A : OS_COLLECTION;
                                 OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_NOT_EQUAL(OS_COL_A,OS_COL_B);
end OS_CCLLECTION_NOT_EQUAL;
pragma INLINE(OS_COLLECTION_NOT_EQUAL);

-- the functions of remove
--

function C_OS_COLLECTION_REMOVE(OS_COL   :  OS_COLLECTION;
                                VALUE    :  U_TYPEPTR) return BOOLEAN;
  pragma INTERFACE(C, c_os_collection_remove);
  pragma INTERFACE_NAME(c_os_collection_remove,
                        C_SUBP_PREFIX & "os_collection_remove");

function OS_COLLECTION_REMOVE(OS_COL   :  OS_COLLECTION;
                              VALUE    :  U_TYPEPTR) return BOOLEAN is
begin
  return C_OS_COLLECTION_REMOVE(OS_COL,VALUE);
end OS_COLLECTION_REMOVE;
pragma INLINE(OS_COLLECTION_REMOVE);
--
-- OS_COLLECTION_REMOVE_FIRST
function C_OS_COLLECTION_REMOVE_FIRST(OS_COL : OS_COLLECTION) return U_TYPEPTR;

  pragma INTERFACE(C, c_os_collection_remove_first);
  pragma INTERFACE_NAME(c_os_collection_remove_first,
                        C_SUBP_PREFIX & "os_collection_remove_first");

function OS_COLLECTION_REMOVE_FIRST(OS_COL : OS_COLLECTION) return U_TYPEPTR is
begin
  return C_OS_COLLECTION_REMOVE_FIRST(OS_COL);
end OS_COLLECTION_REMOVE_FIRST;
pragma INLINE(OS_COLLECTION_REMOVE_FIRST);


--
-- OS_COLLECTION_REMOVE_LAST
function C_OS_COLLECTION_REMOVE_LAST(OS_COL : OS_COLLECTION) return U_TYPEPTR;
```

```
  pragma INTERFACE(C, c_os_collection_remove_last);
  pragma INTERFACE_NAME(c_os_collection_remove_last,
                        C_SUBP_PREFIX & "os_collection_remove_last");

function OS_COLLECTION_REMOVE_LAST(OS_COL : OS_COLLECTION)  return U_TYPEPTR is
begin
  return C_OS_COLLECTION_REMOVE_LAST(OS_COL);
end OS_COLLECTION_REMOVE_LAST;
pragma INLINE(OS_COLLECTION_REMOVE_LAST);

--
-- OS_COLLECTION_ONLY
function C_OS_COLLECTION_ONLY(OS_COL : OS_COLLECTION) return U_TYPEPTR;

  pragma INTERFACE(C, c_os_collection_only);
  pragma INTERFACE_NAME(c_os_collection_only,
                        C_SUBP_PREFIX & "os_collection_only");

function OS_COLLECTION_ONLY(OS_COL : OS_COLLECTION)  return U_TYPEPTR is
begin
  return C_OS_COLLECTION_ONLY(OS_COL);
end OS_COLLECTION_ONLY;
--
-- OS_COLLECTION_ORDERED_EQUAL
function C_OS_COLLECTION_ORDERED_EQUAL(OS_COL_A : OS_COLLECTION;
                                       OS_COLL_B : OS_COLLECTION) return BOOLEAN;

  pragma INTERFACE(C, c_os_collection_ordered_equal);
  pragma INTERFACE_NAME(c_os_collection_ordered_equal,
                        C_SUBP_PREFIX & "os_collection_ordered_equal");


function OS_COLLECTION_ORDERED_EQUAL(OS_COL_A : OS_COLLECTION;
                                     OS_COL_B : OS_COLLECTION) return BOOLEAN is
begin
  return C_OS_COLLECTION_ORDERED_EQUAL(OS_COL_A,OS_COL_B);
end OS_COLLECTION_ORDERED_EQUAL;
pragma INLINE(OS_COLLECTION_ORDERED_EQUAL);
-- the functions of pick
--
-- OS_COLLECTION_PICK
function C_OS_COLLECTION_PICK(OS_COL : OS_COLLECTION) return U_TYPEPTR;

  pragma INTERFACE(C, c_os_collection_pick);
  pragma INTERFACE_NAME(c_os_collection_pick,
                        C_SUBP_PREFIX & "os_collection_pick");

function OS_COLLECTION_PICK(OS_COL : OS_COLLECTION) return U_TYPEPTR is
begin
  return C_OS_COLLECTION_PICK(OS_COL);
end OS_COLLECTION_PICK;
```

```
pragma INLINE(OS_COLLECTION_PICK);
--
--
FUNCTION C_OS_COLLECTION_QUERY(OS_COL     : OS_COLLECTION;
                               ELEMENT    : SYSTEM.ADDRESS;
                               EXPRESSION : SYSTEM.ADDRESS;
                               DB         : DATABASE;
                               FILE_NAME  : OS_INT32;
                               LINE_NUMBER: OS_UNSIGNED_INT32)
                                  RETURN OS_COLLECTION;

  PRAGMA INTERFACE(C, c_os_collection_query);
  PRAGMA INTERFACE_NAME(c_os_collection_query,
                        C_SUBP_PREFIX & "os_collection_query");

FUNCTION OS_COLLECTION_QUERY(OS_COL         : OS_COLLECTION;
                             ELEMENT_TYPE   : STRING;
                             EXPRESS_STRING : STRING;
                             DB             : DATABASE) RETURN OS_COLLECTION IS

-- THIS SAMPLE DEFAULT THAT THE FILE NAME AND LINE NUMBER ARE 0
-- FILE_NAME   : INTEGER := 0;
-- LINE_NUMBER : OS_UNSIGN_INT32 := 0;
ELEMENT_ADDRESS    : SYSTEM.ADDRESS :=
     A_STRINGS.TO_C(A_STRINGS.TO_A(ELEMENT_TYPE(1..ELEMENT_TYPE'LENGTH)));
EXPRESSION_ADDRESS : SYSTEM.ADDRESS :=
     A_STRINGS.TO_C(A_STRINGS.TO_A(EXPRESS_STRING(1..EXPRESS_STRING'LENGTH)));
BEGIN

  RETURN C_OS_COLLECTION_QUERY
       (OS_COL,ELEMENT_ADDRESS,EXPRESSION_ADDRESS,DB,0,0);
END OS_COLLECTION_QUERY;
PRAGMA INLINE(OS_COLLECTION_QUERY);
--
--
FUNCTION C_OS_COLLECTION_QUERY_EXISTS(OS_COL     : OS_COLLECTION;
                                      ELEMENT    : SYSTEM.ADDRESS;
                                      EXPRESSION : SYSTEM.ADDRESS;
                                      DB         : DATABASE;
                                      FILE_NAME  : OS_INT32;
                                      LINE_NUMBER: OS_UNSIGNED_INT32)
                                    RETURN BOOLEAN;

  PRAGMA INTERFACE(C, C_OS_COLLECTION_QUERY_EXISTS);
  PRAGMA INTERFACE_NAME(C_OS_COLLECTION_QUERY_EXISTS,
                        C_SUBP_PREFIX & "os_collection_query_exists");

FUNCTION OS_COLLECTION_QUERY_EXISTS(OS_COL         : OS_COLLECTION;
                             ELEMENT_TYPE   : STRING;
                             EXPRESS_STRING : STRING;
                             DB             : DATABASE) RETURN BOOLEAN IS
```

```
-- THIS SAMPLE DEFAULT THAT THE FILE NAME AND LINE NUMBER ARE 0
-- FILE_NAME   : INTEGER := 0;
-- LINE_NUMBER : OS_UNSIGN_INT32 := 0;
ELEMENT_ADDRESS    : SYSTEM.ADDRESS :=
     A_STRINGS.TO_C(A_STRINGS.TO_A(ELEMENT_TYPE(1..ELEMENT_TYPE'LENGTH)));
EXPRESSION_ADDRESS : SYSTEM.ADDRESS :=
     A_STRINGS.TO_C(A_STRINGS.TO_A(EXPRESS_STRING(1..EXPRESS_STRING'LENGTH)));

BEGIN

  RETURN C_OS_COLLECTION_QUERY_EXISTS
       (OS_COL,ELEMENT_ADDRESS,EXPRESSION_ADDRESS,DB,0,0);
END OS_COLLECTION_QUERY_EXISTS;
PRAGMA INLINE(OS_COLLECTION_QUERY_EXISTS);

--
--
FUNCTION C_OS_COLLECTION_QUERY_PICK(OS_COL     : OS_COLLECTION;
                                    ELEMENT    : SYSTEM.ADDRESS;
                                    EXPRESSION : SYSTEM.ADDRESS;
                                    DB         : DATABASE;
                                    FILE_NAME  : OS_INT32;
                                    LINE_NUMBER: OS_UNSIGNED_INT32)
                                  RETURN U_TYPEPTR;

  PRAGMA INTERFACE(C, C_OS_COLLECTION_QUERY_PICK);
  PRAGMA INTERFACE_NAME(C_OS_COLLECTION_QUERY_PICK,
                        C_SUBP_PREFIX & "os_collection_query_pick");

FUNCTION OS_COLLECTION_QUERY_PICK(OS_COL         : OS_COLLECTION;
                                  ELEMENT_TYPE   : STRING;
                                  EXPRESS_STRING : STRING;
                                  DB             : DATABASE) RETURN U_TYPEPTR IS

-- THIS SAMPLE DEFAULT THAT THE FILE NAME AND LINE NUMBER ARE 0
-- FILE_NAME   : INTEGER := 0;
-- LINE_NUMBER : OS_UNSIGN_INT32 := 0;
ELEMENT_ADDRESS    : SYSTEM.ADDRESS :=
     A_STRINGS.TO_C(A_STRINGS.TO_A(ELEMENT_TYPE(1..ELEMENT_TYPE'LENGTH)));
EXPRESSION_ADDRESS : SYSTEM.ADDRESS :=
     A_STRINGS.TO_C(A_STRINGS.TO_A(EXPRESS_STRING(1..EXPRESS_STRING'LENGTH)));

BEGIN

  RETURN C_OS_COLLECTION_QUERY_PICK
       (OS_COL,ELEMENT_ADDRESS,EXPRESSION_ADDRESS,DB,0,0);
END OS_COLLECTION_QUERY_PICK;
PRAGMA INLINE(OS_COLLECTION_QUERY_PICK);
--
--
```

```
procedure C_OS_COLLECTION_UNION(OS_COL_A : OS_COLLECTION;
                                OS_COL_B : OS_COLLECTION);

  pragma INTERFACE(C, c_os_collection_union);
  pragma INTERFACE_NAME(c_os_collection_union,
                        C_SUBP_PREFIX & "os_collection_union");


procedure OS_COLLECTION_UNION(OS_COL_A : OS_COLLECTION;
                              OS_COL_B : OS_COLLECTION)  is
begin
      C_OS_COLLECTION_UNION(OS_COL_A,OS_COL_B);
end OS_COLLECTION_UNION;
pragma INLINE(OS_COLLECTION_UNION);

--------------------------------------------------------------------------------
-- os_cursors
--------------------------------------------------------------------------------

function C_OS_CURSOR_CREATE(OS_COLL : OS_COLLECTION; B:OS_BOOLEAN)
                         return  OS_CURSOR;
   pragma INTERFACE(C, c_os_cursor_create);
   pragma INTERFACE_NAME(c_os_cursor_create,
                         C_SUBP_PREFIX & "os_cursor_create");

function OS_CURSOR_CREATE(OS_COLL : OS_COLLECTION) return  OS_CURSOR is
UPDATED : boolean := false;
begin
    return C_OS_CURSOR_CREATE(OS_COLL, B_TO_OSB(UPDATED));
end OS_CURSOR_CREATE;
pragma INLINE(OS_CURSOR_CREATE);

procedure C_OS_CURSOR_DELETE(OS_CUR : OS_CURSOR);
   pragma INTERFACE(C, c_os_cursor_delete);
   pragma INTERFACE_NAME(c_os_cursor_delete,
                         C_SUBP_PREFIX & "os_cursor_delete");
procedure   OS_CURSOR_DELETE(OS_CUR : OS_CURSOR) is
begin
      C_OS_CURSOR_DELETE(OS_CUR);
end OS_CURSOR_DELETE;
pragma INLINE(OS_CURSOR_DELETE);


function C_OS_CURSOR_FIRST(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_first);
   pragma INTERFACE_NAME(c_os_cursor_first,
                         C_SUBP_PREFIX & "os_cursor_first");
function OS_CURSOR_FIRST(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
   return C_OS_CURSOR_FIRST(OS_CUR);
end OS_CURSOR_FIRST;
```

```
pragma INLINE(OS_CURSOR_FIRST);


function C_OS_CURSOR_MORE(OS_CUR : OS_CURSOR) return BOOLEAN;
   pragma INTERFACE(C, c_os_cursor_more);
   pragma INTERFACE_NAME(c_os_cursor_more,
                         C_SUBP_PREFIX & "os_cursor_more");
function OS_CURSOR_MORE(OS_CUR : OS_CURSOR) return boolean is
begin
  return C_OS_CURSOR_MORE(OS_CUR);
end OS_CURSOR_MORE;
pragma INLINE(OS_CURSOR_MORE);


function C_OS_CURSOR_NEXT(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_next);
   pragma INTERFACE_NAME(c_os_cursor_next,
                         C_SUBP_PREFIX & "os_cursor_next");

function  OS_CURSOR_NEXT(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
  return C_OS_CURSOR_NEXT(OS_CUR);
end OS_CURSOR_NEXT;
pragma INLINE(OS_CURSOR_NEXT);

end OS_COLLECTION_PKG;
```

C.10 Interface Program: os_cur.a

```
-- Basic cursor interface to ObjectStore from the Ada programming
-- language prototype  implementation by Li Chou
--

with OSTORE; use OSTORE;
with OS_TYPES; use OS_TYPES;
generic
  type U_TYPE is private;
  type U_TYPEPTR is access U_TYPE;

package OS_CURSOR_PKG is
-- cursor's functions

function OS_CURSOR_CREATE(OS_COLL : OS_COLLECTION;
                          UPDATED  : BOOLEAN := false ) return  OS_CURSOR;

procedure OS_CURSOR_COPY(OS_CUR_A : OS_CURSOR;
                         OS_CUR_B : OS_CURSOR);

procedure OS_CURSOR_DELETE(OS_CUR : OS_CURSOR);

function OS_CURSOR_FIRST(OS_CUR : OS_CURSOR) return U_TYPEPTR;

procedure OS_CURSOR_INSERT_AFTER(OS_COL  :  OS_CURSOR;
                                 VALUE   :  U_TYPEPTR);

procedure OS_CURSOR_INSERT_BEFORE(OS_COL  :  OS_CURSOR;
                                  VALUE   :  U_TYPEPTR);

function OS_CURSOR_LAST(OS_CUR : OS_CURSOR) return U_TYPEPTR;

function OS_CURSOR_MORE(OS_CUR : OS_CURSOR) return BOOLEAN;

function OS_CURSOR_NEXT(OS_CUR : OS_CURSOR) return U_TYPEPTR;

function OS_CURSOR_NULL(OS_CUR : OS_CURSOR) return boolean;

function OS_CURSOR_PREVIOUS(OS_CUR : OS_CURSOR) return U_TYPEPTR;

procedure OS_CURSOR_REMOVE_AT(OS_CUR : OS_CURSOR);

function OS_CURSOR_RETRIEVE(OS_CUR : OS_CURSOR) return U_TYPEPTR;

function OS_CURSOR_VALID(OS_CUR : OS_CURSOR) return boolean;

end OS_CURSOR_PKG;
```

C.11 Interface Program: os_cur_b.a

```
-- Basic collection interface to ObjectStore from the Ada programming
-- language prototype  implementation by Li Chou
--
with LANGUAGE; use LANGUAGE;
with OSTORE; use OSTORE;
with OS_TYPES; use OS_TYPES;

package body OS_CURSOR_PKG is

-------------------------------------------------------------------------
-- os_cursors
-------------------------------------------------------------------------

function C_OS_CURSOR_CREATE(OS_COLL : OS_COLLECTION;
                            B : OS_BOOLEAN) return  OS_CURSOR;
   pragma INTERFACE(C, c_os_cursor_create);
   pragma INTERFACE_NAME(c_os_cursor_create,
                         C_SUBP_PREFIX & "os_cursor_create");

function OS_CURSOR_CREATE(OS_COLL : OS_COLLECTION;
                          UPDATED : boolean := false ) return  OS_CURSOR is

begin
    return C_OS_CURSOR_CREATE(OS_COLL, B_TO_OSB(UPDATED));
end OS_CURSOR_CREATE;
pragma INLINE(OS_CURSOR_CREATE);
-------------------------------------------------------------------
--
procedure C_OS_CURSOR_COPY(OS_CUR_A : OS_CURSOR;
                           OS_CUR_B : OS_CURSOR);
   pragma INTERFACE(C, c_os_cursor_copy);
   pragma INTERFACE_NAME(c_os_cursor_copy,
                         C_SUBP_PREFIX & "os_cursor_copy");
procedure OS_CURSOR_COPY(OS_CUR_A : OS_CURSOR;
                         OS_CUR_B : OS_CURSOR)  is
begin
   C_OS_CURSOR_COPY(OS_CUR_A, OS_CUR_B);
end OS_CURSOR_COPY;
pragma INLINE(OS_CURSOR_COPY);
-------------------------------------------------------------------
--
procedure C_OS_CURSOR_DELETE(OS_CUR : OS_CURSOR);
   pragma INTERFACE(C, c_os_cursor_delete);
   pragma INTERFACE_NAME(c_os_cursor_delete,
                         C_SUBP_PREFIX & "os_cursor_delete");
procedure   OS_CURSOR_DELETE(OS_CUR : OS_CURSOR) is
begin
      C_OS_CURSOR_DELETE(OS_CUR);
end OS_CURSOR_DELETE;
```

```
pragma INLINE(OS_CURSOR_DELETE);
------------------------------------------------------------
--
function C_OS_CURSOR_FIRST(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_first);
   pragma INTERFACE_NAME(c_os_cursor_first,
                         C_SUBP_PREFIX & "os_cursor_first");
function OS_CURSOR_FIRST(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
   return C_OS_CURSOR_FIRST(OS_CUR);
end OS_CURSOR_FIRST;
pragma INLINE(OS_CURSOR_FIRST);
------------------------------------------------------------
--
procedure C_OS_CURSOR_INSERT_AFTER(OS_COL   :  OS_CURSOR;
                                   VALUE    :  U_TYPEPTR);
  pragma INTERFACE(C, c_os_cursor_insert_after);
  pragma INTERFACE_NAME(c_os_cursor_insert_after,
                        C_SUBP_PREFIX & "os_cursor_insert_after");

procedure OS_CURSOR_INSERT_AFTER(OS_COL   :  OS_CURSOR;
                                 VALUE    :  U_TYPEPTR) is
begin
  C_OS_CURSOR_INSERT_AFTER(OS_COL,VALUE);
end OS_CURSOR_INSERT_AFTER;
pragma INLINE(OS_CURSOR_INSERT_AFTER);
------------------------------------------------------------
--
procedure C_OS_CURSOR_INSERT_BEFORE(OS_COL   :  OS_CURSOR;
                                    VALUE    :  U_TYPEPTR);
  pragma INTERFACE(C, c_os_cursor_insert_before);
  pragma INTERFACE_NAME(c_os_cursor_insert_before,
                        C_SUBP_PREFIX & "os_cursor_insert_before");

procedure OS_CURSOR_INSERT_BEFORE(OS_COL   :  OS_CURSOR;
                                  VALUE    :  U_TYPEPTR) is
begin
  C_OS_CURSOR_INSERT_BEFORE(OS_COL,VALUE);
end OS_CURSOR_INSERT_BEFORE;
pragma INLINE(OS_CURSOR_INSERT_BEFORE);
------------------------------------------------------------
--
function C_OS_CURSOR_LAST(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_last);
   pragma INTERFACE_NAME(c_os_cursor_last,
                         C_SUBP_PREFIX & "os_cursor_last");
function OS_CURSOR_LAST(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
   return C_OS_CURSOR_LAST(OS_CUR);
end OS_CURSOR_LAST;
pragma INLINE(OS_CURSOR_LAST);
```

```
-------------------------------------------------------------
--
function C_OS_CURSOR_MORE(OS_CUR : OS_CURSOR) return BOOLEAN;
   pragma INTERFACE(C, c_os_cursor_more);
   pragma INTERFACE_NAME(c_os_cursor_more,
                         C_SUBP_PREFIX & "os_cursor_more");
function OS_CURSOR_MORE(OS_CUR : OS_CURSOR) return boolean is
begin
  return C_OS_CURSOR_MORE(OS_CUR);
end OS_CURSOR_MORE;
pragma INLINE(OS_CURSOR_MORE);
-------------------------------------------------------------
--
function C_OS_CURSOR_NEXT(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_next);
   pragma INTERFACE_NAME(c_os_cursor_next,
                         C_SUBP_PREFIX & "os_cursor_next");

function  OS_CURSOR_NEXT(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
  return C_OS_CURSOR_NEXT(OS_CUR);
end OS_CURSOR_NEXT;
pragma INLINE(OS_CURSOR_NEXT);
-------------------------------------------------------------
--
function C_OS_CURSOR_NULL(OS_CUR : OS_CURSOR) return BOOLEAN;
   pragma INTERFACE(C, c_os_cursor_null);
   pragma INTERFACE_NAME(c_os_cursor_null,
                         C_SUBP_PREFIX & "os_cursor_null");
function OS_CURSOR_NULL(OS_CUR : OS_CURSOR) return boolean is
begin
  return C_OS_CURSOR_NULL(OS_CUR);
end OS_CURSOR_NULL;
pragma INLINE(OS_CURSOR_NULL);
-------------------------------------------------------------
--
function C_OS_CURSOR_PREVIOUS(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_previous);
   pragma INTERFACE_NAME(c_os_cursor_previous,
                         C_SUBP_PREFIX & "os_cursor_previous");
function OS_CURSOR_PREVIOUS(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
   return C_OS_CURSOR_PREVIOUS(OS_CUR);
end OS_CURSOR_PREVIOUS;
pragma INLINE(OS_CURSOR_PREVIOUS);
-------------------------------------------------------------
--
procedure C_OS_CURSOR_REMOVE_AT(OS_CUR : OS_CURSOR);
   pragma INTERFACE(C, c_os_cursor_remove_at);
   pragma INTERFACE_NAME(c_os_cursor_remove_at,
                         C_SUBP_PREFIX & "os_cursor_remove_at");
```

```
procedure OS_CURSOR_REMOVE_AT(OS_CUR : OS_CURSOR)  is
begin
   C_OS_CURSOR_REMOVE_AT(OS_CUR);
end OS_CURSOR_REMOVE_AT;
pragma INLINE(OS_CURSOR_REMOVE_AT);
-------------------------------------------------------------
--
function C_OS_CURSOR_RETRIEVE(OS_CUR : OS_CURSOR) return U_TYPEPTR;
   pragma INTERFACE(C, c_os_cursor_retrieve);
   pragma INTERFACE_NAME(c_os_cursor_retrieve,
                         C_SUBP_PREFIX & "os_cursor_retrieve");
function OS_CURSOR_RETRIEVE(OS_CUR : OS_CURSOR) return U_TYPEPTR is
begin
   return C_OS_CURSOR_RETRIEVE(OS_CUR);
end OS_CURSOR_RETRIEVE;
pragma INLINE(OS_CURSOR_RETRIEVE);
-------------------------------------------------------------
--
function C_OS_CURSOR_VALID(OS_CUR : OS_CURSOR) return BOOLEAN;
   pragma INTERFACE(C, c_os_cursor_valid);
   pragma INTERFACE_NAME(c_os_cursor_valid,
                         C_SUBP_PREFIX & "os_cursor_valid");
function OS_CURSOR_VALID(OS_CUR : OS_CURSOR) return boolean is
begin
  return C_OS_CURSOR_VALID(OS_CUR);
end OS_CURSOR_VALID;
pragma INLINE(OS_CURSOR_VALID);

end OS_CURSOR_PKG;
```

C.12 Interface Program: except.a

```
--Exceptions for ObjectStore/Ada interface.
-- Basic interface to ObjectStore from the Ada programming language
-- Prototype design and implementation by Dave Rosenberg of Object
-- Design, Inc.

with SYSTEM; use SYSTEM;
with LANGUAGE; use LANGUAGE;
with OS_TYPES; use OS_TYPES;
package OS_EXCEPTIONS is

LAST_EXCEPTION: constant INTEGER := 0;
subtype OS_EXCEPTION_INDEX is OS_INT32 range 0 .. LAST_EXCEPTION;
procedure OS_ADA_EXCEPTION(ERR: OS_EXCEPTION_INDEX);
pragma EXTERNAL_NAME(OS_ADA_EXCEPTION,
     C_SUBP_PREFIX & "os_ada_exception");

ERR_ADDRESS_SPACE_FULL: EXCEPTION;


end OS_EXCEPTIONS;
```

C.13 Interface Program: except_b.a

```
--Exceptions for ObjectStore/Ada interface.
-- Basic interface to ObjectStore from the Ada programming language
-- Prototype design and implementation by Dave Rosenberg of Object
-- Design, Inc.

package body OS_EXCEPTIONS is

procedure OS_ADA_EXCEPTION(ERR: OS_EXCEPTION_INDEX) is
begin
case ERR is
  when 0 => raise ERR_ADDRESS_SPACE_FULL;
  when others => null;
end case;
end OS_ADA_EXCEPTION;

end OS_EXCEPTIONS;
```

Bibliography

1. Ada Joint Porgram Office, DoD. *Ada Reference Manual, ANJI/MIL-STD-1815A*, January 1983.
2. Atkinson, M. P., et al. "An Approach to Persistent Programming," *The Computer Journal*, *26*(4):360–365 (1983).
3. Atkinson, M. P., et al. *Data Types and Persistence*. Berlin: Springer-Verlag, 1988.
4. AT&T. *Unix System V AT&T C++ Language System*, 1989. selected Code 307-144.
5. Berre, Arne J. and T. Lougenia Anderson. "The HyperModel Benchmark for Evaluating Object-Oriented Databases." *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD* chapter 5, 75–91, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991.
6. Booch, Grady. *Software Engineering with Ada*. Benjamin/Cummings Publishing Company, Inc., 1986.
7. Booch, Grady. *Object Oriented Design with applications*. Benjamin/Cummings Publishing Company, Inc., 1991.
8. Cardelli, Luca and David MacQueen. "Persistence and Type Abstraction." *Data Types and Persistence* edited by M. P. Atkinson, et al., Berlin: Springer-Verlag, 1988.
9. Cattel, R.G.G. "Object-Oriented DBMS Performance Measurement." *Proceedings of the 2nd Workshop on OODBS*. 364–367. 1988.
10. Harper, Robert. "Modules and Persistence in Standart ML." *Data Types and Persistence* edited by M. P. Atkinson, et al., Berlin: Springer-Verlag, 1988.
11. Jacobs, Captain Timothy M. *An Object-Oriented Database Implementation of The Magic VSLI Layout Design System*. MS thesis, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1991.
12. Johnson, Eric F. and Kevin Reichard. "The X window Application Programming," *Portland: MIS Press* (1989).
13. Klabunde, Gary Wayne. *An Animated Graphical Postprocessor for the Saber Wargame*. MS thesis, AFIT/GCS/ENG/91D-10, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1991.
14. Lamb, Charles, et al. "The ObjectStore Database System," *Communications of the ACM*, *34*(10):50–64 (1989).
15. Leopold, Vince. "Object-Oriented Programming in Ada, A Viable Method," *IEEE, NAECON 89'*, *2*:549–552 (1989).
16. Neville, Donna and Dit Morse. *Pro*Ada User's Guide*. Oracle Corporation, November 1986.
17. Object Design, Inc., Burlington, Massachusetts. *ObjectStore User Guide* (1.1 Edition), 1991.
18. Rosenberg, Dave. *ObjectStore and Ada*. Object Design, Inc., Burlington, Massachusetts, January 1992.
19. Rumbaugh, James, et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
20. Scheifler, Robert W. and Gettys Jim. "The X window system," *ACM Transaction on Graphics*, 5:79–109 (April 1986).
21. Schonberg, Edmond. "Contrasts : Ada 9X and C++," *CrossTalk* (September 1992).

22. Thatte, Satish. "Persistent Memory: Merging AI-knowledge and Databases." *Readings in Object Oriented Database Systems* edited by Stanley B. Zdonik and David Maier, 242–250, Morgan Kaufmann Publishers, 1990.

23. Unisys Corporation, 12010 Sunrise Valley Drive. *Ada Interfaces to X Window System*, March 1989. Contract No. F19628-88-D-0031.

24. Verdix Corporation. *Verdix Ada Development System*, 1991.

Vita

Lt Col Li Chou (ROCAF, Taiwan) was born on 28 October 1957 in I-Lan, Taiwan, Republic of China. He graduated from high school in I-Lan in June, 1975. He then entered the National Defense Medical Center from which he graduated in 1980 with a Bachelor Degree in Pharmacy and a commission as a Lieutenant in the Republic of China Air Force (ROCAF). His first duty assignment was at the Fifth Medical Corps, Tau-Yan, Taiwan as a pharmacy officer. In early 1983 he transferred to the 816th Regional Hospital as a pharmacy officer. He received his promotion to Captain in August, 1983. In the middle of 1984 he relocated to the Surgeon General, Headquarters, ROCAF, and joined the Medical Administration Staff. While there, he entered Central University for a training course in Computer Science in August, 1987. He got his promotion to Major during his training, in January 1988. He entered the School of Engineering, Air Force Institute of Technology in May, 1991. During his staying here, he received his promotion to Lt Col in January, 1992.

Permanent address: 4F, No. 15-2, Ln 89, Shih-Tong Rd,
Shih-Ling, Taipei, Taiwan, R.O.C.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1993	Master's Thesis

4. TITLE AND SUBTITLE
OBJECT-ORIENTED DATABASE ACCESS FROM ADA

5. FUNDING NUMBERS

6. AUTHOR(S)
Li Chou

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/93M-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Phil Lienert
ASD/RWWW Wright-Patterson AFB, OH 45433 x53969
Mr. Joseph V. Giordano
Rome Laboratory RL/C3AB, Griffiss AFB, NY 13441-5700 (DSN) 587-2805

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Ada embodies many modern software engineering principles, namely, modifiability, efficiency, reliability, and understandability. Its powerful data abstraction allows programmers to easily model objects in the real world. A database management system (DBMS) provides long term storage. It provides a convenient and efficient environment to manipulate data. Currently, with Ada access to a DBMS is typically done through the use of a language extension and a preprocessor to convert the extensions to library calls appropriate for the DBMS. However, these systems are limited on more complex applications, such as computer-aided engineering design. Object-oriented design (OOD) is a new way of thinking about problems using models organized around real-world concepts. Object-oriented database management systems (OODBMS), include most benefits of relational DBMS (RDBMS) and, in addition, provide the capability to manipulate complex, heterogeneous data. ObjectStore is an OODBMS. This thesis describes an interface from Ada to ObjectStore which could fulfill the requirements of complex applications. Our Ada/ObjectStore interface performed better in CPU time than the supplied C/ObjectStore interface. However, overall there is not much difference between Ada/ObjectStore and C/ObjectStore. It is clear that Ada/ObjectStore provides the capability of data persistence to Ada. This result favorably affects program length, program development time, program maintainability, and application reliability.

14. SUBJECT TERMS
Ada, C, Object-oriented database management system, ObjectStore

15. NUMBER OF PAGES
158

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

*U.S.GPO:1990-0-273-271